A Graphics Processing Unit Based Method for Dynamic Real-Time Global Illumination

Insu Yu

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of the

University College London.

2011

Dedicated to my wife, *Yoon Jung* and my son, *Ian*

I, *Insu Yu*, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Insu Yu

Abstract

Real-time realistic image synthesis for virtual environments has been one of the most actively researched areas in computer graphics for over a decade. Images that display physically correct illumination of an environment can be simulated by evaluating a multi-dimensional integral equation, called the *rendering equation*, over the surfaces of the environment. Many *global illumination* algorithms such as path-tracing, photon mapping and distributed ray-tracing can produce realistic images but are generally unable to cope with dynamic lighting and objects at interactive rates. It still remains one of most challenging problems to simulate physically correctly illuminated dynamic environments without a substantial pre-processing step.

In this thesis we present a rendering system for dynamic environments by implementing a customized rasterizer for global illumination entirely on the graphics hardware, the *Graphical Processing Unit*. Our research focuses on a parameterization of discrete visibility field for efficient indirect illumination computation. In order to generate the visibility field, we propose a CUDA-based (*Compute Unified Device Architecture*) rasterizer which builds *Layered Hit Buffers* (LHB) by rasterizing polygons into multi-layered structural buffers in parallel. The LHB provides a fast visibility function for any direction at any point. We propose a cone approximation solution to resolve an aliasing problem due to limited directional discretization. We also demonstrate how to remove structure noises by adapting an interleaved sampling scheme and discontinuity buffer. We show that a gathering method amortized with a multi-level Quasi Mont Carlo method can evaluate the rendering equation in real-time.

The method can realize real-time walk-through of a complex virtual environment that has a mixture of diffuse and glossy reflection, computing multiple indirect bounces on the fly. We show that our method is capable of simulating fully *dynamic environments* including changes of view, materials, lighting and objects at interactive rates on commodity level graphics hardware.

Acknowledgements

I am extremely grateful to my supervisor, Prof. Mel Slater and second supervisor, Dr. Jan Kautz for invaluable guidance. I would also thank to fellow researchers: Jesper Mortensen, Pankaj Khanna and especially Joel Jordan for encouragement. This work was funded by EPSRC projects (EP/C511824/1 and GR/R13685/01).

Finally, I would like to thank my parents and parents-in-law for their love, unconditional support and pray during my PhD.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Computer Graphics has changed significantly in the last three decades. Prior to the 1980s, the simulation of light was extremely limited to local illumination models. Most of the major theoretical advances in simulating realistic lighting in 3D scenes were made between the 1980s and 1990s. During these decades, two new innovative concepts were introduced within the computer graphics community; one based on a point sampling *Ray Tracing* method, and the other is a finite element method such as a *Radiosity* solution. Ray Tracing techniques tend to solve the rendering equation per pixel over an image plane using the Monte Carlo integration method. In contrast, the finite element method computes the radiosity value for every element in the scene.

Both of these concepts were already employed in different fields; for example ray tracing was a popular method in optics, whereas the radiosity notion was widely used in heat transfer problems. These methods both make extensive use of the physical nature of light, which is related to the goal of computer graphics to produce photo realistic images based on the physical phenomena of light interaction. *Global Illumination* takes into account not only the light that comes directly from light sources, known as *Local Illumination*, but also further contributions from light bounces, in which the light rays are reflected by other surfaces in the scene. Images rendered using global illumination appear more photorealistic than images simply generated by local illumination models.

Although many studies have focused on improving the rendering speed while maintaining quality, existing approaches are usually limited to computing only a part of the global illumination effect for simple dynamic scenes. It is impractical to make a real-time path tracer due to the complexity of the rendering equation. However, in recent years, advanced graphics hardware is capable of providing flexible rendering pipelines, so that global illumination have become more practical and realistic. This research belongs to that category of study, which enhances the rendering speed of path tracing in the context of global illumination. The main issue that is addressed in this research is massively improving the indirect illumination computation. The key problem is to how to solve the visibility computation efficiently for dynamic scenes. In this thesis, an efficient multi-layered visibility structure is presented by implementing a customized rasterizer. The visibility field provides instant access to visibility queries without the need for computing the ray-polygon intersections. We present a GPU-based Monte Carlo path-tracing

for dynamic environments, which achieves a real-time path tracer entirely on the graphic hardware, such that global illumination is well perceived without loss of realism.

1.2 Motivation

Computer graphics rendering may be expressed as a technique to simulate all reflections of light and reproduce the accurate intensity of the light at any given point. However, some computer graphics algorithms use partial solutions of the rendering equation in order to achieve interactive frame rates. Figure 1.1 shows an example of globally illuminated scene, which represents many natural lighting phenomena. Two rendered images of the Sponza model are shown; one rendered with direct lighting only, and the other using path tracing. The right-hand image, which was rendered using global illumination, appears to be more realistic than the left-hand image, which was rendered using a direct illumination algorithm.

Figure 1.1: Comparison of Direct and Indirect Illumination of Sponza Model.

The fundamental goal of this research is to provide a rendering system that handles complex dynamic elements, whilst allowing updating of global illumination on the fly without any pre-processing step. In order to achieve this, graphics card hardware is exploited to generate a Monte Carlo path tracer for real-time rendering of dynamic scenes. One of the most expensive components of global illumination is the visibility determination, and global illumination solutions for multiple inter-reflection of light in a dynamic environment require the computation of a global illumination solution for each frame. An acceleration structure improve ray-polygon intersection computation for complex scenes, but it becomes very costly to rebuild an accelerations structure per frame which often prohibits real-time rendering. This results in the motivation to build a new acceleration structure to speed up indirect illumination computation. This acceleration structure is based on the idea of scarifying accuracy of visibility in order to achieve a highly interactive rate, without losing perceived realism for dynamic environments. The perceptual influence of visibility approximations on indirect illumination has been studied in order to evaluate whether inaccurate visibility approximations are perceived as realistic as the reference rendering. The following section explains the user study in more detail.

1.2.1 Approximated Visibility for Indirect Illumination

Global illumination effects, such as indirect illumination, are known to be perceptually important, but are often omitted or coarsely approximated due to their high rendering cost, especially in interactive applications. One of the most expensive components in global illumination is visibility determination, where it must be decided whether two points are mutually visible or not. This is usually performed accurately using the ray-casting method, and there are many methods that exist to speed up accurate visibility queries. However, it was unknown whether accurate visibility for indirect illumination is perceptually important at all.

(a) Imperfect (25%) (b) Imperfect (50%) (c) Imperfect (75%)

(d) Amb. Occl. ($r = 0.05$) (e) Amb. Occl. ($r = 0.10$) (f) Amb. Occl. ($r = 0.20$)

(g) Dir. Amb. Occl. ($r = 0.05$) (h) Dir. Amb. Occl. ($r = 0.10$) (i) Reference

Figure 1.2: Renderings of the arches scene, where the indirect illumination in each image is computed with a different visibility approximation. A psychophysical study shows that many of these visibility approximations produce images that are perceptually very similar to reference renderings.

In our latest perception study [YCK+09], the use of approximated visibility is evaluated for efficient global illumination, in order to determine the relationship between the realism of rendered images and the accuracy of the visibility function. Traditionally, the accurate visibility function is used for light transport. However, the indirect illumination that is perceived on a daily basis is rarely of a high

frequency nature. This is because the most significant aspect of light transport in real world scenes is diffuse, therefore displaying a smooth gradation. In [YCK+09], a psychophysical study is conducted on the perceptual influence of approximate visibility on indirect illumination, where the perceptual influence of different visibility approximations is initially determined by carrying out a series of psychophysical experiments. The data is then analyzed to evaluate how different approximated solutions affect the perceived realism of rendering under global illumination. The formal study uses global illumination scenes as shown in Figure 1.2 rendered with different approximations, such as imperfect visibility [RGK+08], ambient occlusion [ZIK98] and directional ambient occlusion [SGNS07]. It shows different renderings of an arches scene, where the reference uses the accurate visibility function, and the other ones are computed using a visibility approximation for indirect illumination. A psychophysical analysis was performed on the data in order to determine which visibility approximations are perceptually acceptable.

The experiments show that using certain visibility approximations yields results that are perceptually very similar to reference renderings. In other words, visibility approximations can be used in global illumination while maintaining an appearance that is perceptually similar to a reference solution. Furthermore, many visibility approximations yield renderings that are perceived to be realistic despite perceptible differences to reference renderings. The perception study therefore validates the use of visibility approximations in previous works [RGK+08, REG+09]. The proposed real-time rendering solution in this research stems from this notion that using visibility approximations to compute fast indirect illumination can result in renderings that are considered to be as realistic as accurate solutions.

1.3 Scope and Limitation

The proposed solution supports real-time path tracing [Kaj86] for global illumination. A number of simplifying assumptions are made, which are applied throughout the thesis and described below:

- Wavelength independence: It is assumed that there is no interaction between different wavelengths. A few samples of RGB wavelengths will be taken independently, in order to estimate the radiance value at a point.

- Time invariance: It is assumed that the entire solution for the distribution of energy will remain the same over time. There will be no time delay in emitting the energy that is gathered in a particular moment.

- Non-participating medium: One of most significant simplifying assumptions is that light travels in a vacuum space. In other words, this research will only consider a non-participating medium. Therefore it is assumed that light interaction will only occur on surfaces.

- Polygonal based objects are used: Although the approach in this research could be extended to parametric surfaces, this study is limited to the use of polygonal surfaces for simplicity.

It is demonstrated that this solution can handle physically correct global illumination of over hundreds of thousands of polygons at a real-time rate. Although this implementation is limited to scenes containing mainly diffuse materials, extensions for glossy reflection could be achieved by increasing the

sampling rates when gathering the directional samples. However, this solution is not ideal for specular or high glossy surfaces and suffers from a ghost effect for a perfect specular surface, due to the limitation of discrete representation of the SLHB. In addition, transparency is not considered in this dissertation. Our method deals well with dynamic environments, including those incorporating changes of view, materials, lighting and objects. Dynamic objects with over hundreds of thousands of polygons interacting with a few bounces of indirect illumination can be achieved in real-time. This solution can cope with various lighting environments, such as point, directional and area light sources. On top of this, it can easily handle low frequency lighting as well as complex environment lighting conditions, which would take far too long to compute with many other global illumination solutions. The proposed visibility data structure can be extended to support instant radiosity solutions [Kel97]. The method was implemented in Compute Unified Device Architecture (CUDA) to maximize the graphics hardware when building visibility structures and propagating energy in parallel processing. The proposed GPU Monte Carlo path tracing method targets the applications for which a real time rendering is essential for complex scenes with plausible realism. So it does not aim for high-end visualisations such as movies but for real time rendering with global illumination.

1.4 Contribution

The overall contribution of this thesis is the development of a new GPU based rendering system, from GPU rasterization to a real-time path tracing solution using a proposed acceleration structure, entirely implemented on a CUDA-based platform. Our proposed rasterization method utilizes graphics hardware to build a visibility field for instant access.

The three key contributions are:

- A CUDA-based rasterizer: This research has proposed a new and efficient CUDA Deep Rasterization method. Conventional OpenGL (or Direct 3D) based rasterization has the limitation of constructing the structural fragment buffer output in a single-pass, whereas the new method is able to rasterize objects into a customized multi-layered structural buffers. The performance is comparable to OpenGL rendering, while providing fully customizable pipeline and output.

- Spherical Layered Hit Buffers(SLHB): The thesis presents an efficient parameterization of the visibility field in a discrete manner, which provides an acceleration structure for instant occlusion query. The proposed CUDA rasterizer is used to build multi-layered hit buffers in order to maintain the visibility structure. For dynamic scenes, a SLHB is built instantly in every frame.

- A GPU-based Monte-Carlo path tracer: The research proposes deterministic and stochastic gathering methods that utilize a spherical layered hit buffers for real-time rendering. The Monte-Carlo path tracer is capable of computing multiple indirect bounces on the fly.

1.5 Organization of the Thesis

This thesis is organized as follows:

The first chapter states the problem and the motivation behind the research, together with the scope of the study and the main contributions.

Chapter 2 follows with an extensive literature review on the theory of a number of rendering techniques related to this work. The fundamental theory of global illumination is presented, followed by an introduction to the early global illumination solutions. The literature review focuses mainly on interactive and real-time global illumination, alongside the latest works on GPU based approaches. In addition to this, our earlier research works will be discussed, regarding the Virtual Light Field(VLF) [SMKY04], which uses pre-computed light fields for real-time rendering. Although the VLF approach is capable of providing a walk-through in real-time, it is limited to static scenes due to the high cost of rebuilding the light field. The new rendering solution determined in this research is then introduced, which overcomes the dynamic problems.

Chapter 3 introduces the fundamental data structure of a new rendering solution to construct a discrete visibility field for instant accessing of occlusion information.

In Chapter 4, a CUDA-based rasterizer is presented, which is capable of writing structural multi-layered depths buffers. Chapter 5 goes on to explain how to utilise the GPU to accelerate the rendering system and demonstrates that the path-tracing method can be achieved in real-time. Implementation details are discussed, to show that a real-time rendering of global illumination models is achievable by using the Spherical Layered Hit Buffers(a pre-computed visibility set).

The results from the new rendering method are presented in Chapter 6. Finally, Chapter 7 summarizes the proposed rendering algorithms, which is followed by a brief discussion of proposed future research. Appendix A includes the notation and symbols used in this thesis, and Appendix B gives a list of papers published in the course of developing this research.

Chapter 2

Overview of Global Illumination

This chapter begins with introducing the basic terms and definitions required to formulate the global illumination problem in mathematical form. Some notations used to describe the models in geometric form are described, followed by discussion of the concept of radiometry to describe light in the computer graphics field. A fundamental mathematical equation describes the light transport in a three-dimensional environment, which is reviewed to explain the complexity of the physical presentation of light. This equation is derived in a linear transport operator form so that the integral form can be solved in a recursive manner. An overview of the global illumination solution to the rendering equation is also reviewed. This is considered in three different categories, which are ray tracing, finite element radiosity and hybrid approaches. Some works in Image Based Rendering are also reviewed, in the context of the global illumination problem. Due to the nature of multiple integrals in the rendering equation, it has been very difficult to solve global illumination in real-time. However, many attempts have been made to achieve this goal by exploiting graphics hardware. An overview of graphics hardware solutions is presented to observe the latest developments using the parallel nature of the computationally intensive procedures in computer graphics. The last part of this chapter addresses this research group's early study on real-time walkthrough of globally illuminated scenes for static environments.

2.1 Fundamentals of Illumination Theory

The primary goal of this section is to establish the terminology necessary to understand illumination theory. Several physical quantities are required to express the physical definition of global illumination. The notation used in this section is explained in Appendix A.

2.1.1 Models of Light

Light is electromagnetic radiation, which carries energy and momentum with a wavelength that is visible to the eye. Three basic properties of light are *Intensity* (or amplitude), which is the human perception of brightness of light; *Frequency* (or wavelength), which is the color of the light; and *Polarization* (or angle or vibration), which accounts for vibration in a multitude of directions. The physics of light is often explained in several models:

- Quantum optics: The foundation for interaction of light and medium, where the behavior of light is explained at the submicroscopic level.

- Wave Optics: Describes light as electromagnetic waves, which can model interference and diffraction.
- Geometrical Optics: Describes light as a independent ray that follows geometric rules. In this model, light is emitted, reflected and transmitted.

In this thesis, the geometrical optics model is used as the fundamental model of light. This means that some effects, such as diffraction, interference and polarization are ignored. The transmission of light through participating media is also ignored. Another assumption is that light has infinite speed, such that the light energy can reach the status of the equilibrium distribution immediately.

2.1.2 Geometry

Surface and Direction

Surface points are points within the three dimensional space that describe positions on the surface of an object. The set of all surface points is denoted by A. A surface normal to the object surface denoted as N_x can be constructed at any surface point x. The differential surface area around point x is written as dA_x. Spherical coordinates are often used to illustrate a direction in the hemisphere. A direction is identified by two angles, (θ, ϕ). Ψ is used to denote the *incoming (incident) direction* and Θ is used for *outgoing direction*. The hemisphere, containing all directions, is denoted by Ω. A set of all directions on hemispheres at point x is denoted as Ω_x (more geometric symbols are explained in Appendix A).

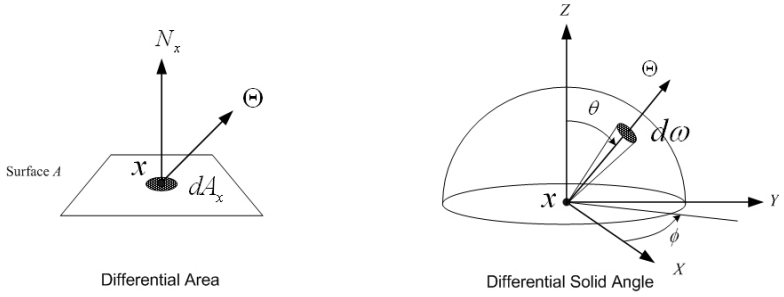


Figure 2.1: Geometry Notations.

Differential Solid Angle

The description of energy exchanges in geometry requires the notion of a *solid angle*, which is used to measure the area of the projection of an object onto the unit sphere as seen from a point. This describes how big the object appears to be from that point. The solid angle is expressed in *steradians (sr)*. The solid angle subtended by the whole sphere $\Omega_{4\pi}$ is 4π sr, which is the entire area of a unit sphere. Therefore, the solid angle extended by a hemisphere Ω is 2π sr.

$$d\omega = \sin\theta \, d\theta \, d\phi \tag{2.1}$$

A differential solid angle ($d\omega$) around a direction (Θ or (θ, ϕ)) is expressed by considering the differential area on the unit sphere. The size of a differential solid angle in spherical coordinates is calculated in Equation 2.1.

2.1.3 Radiometry

Radiometry is the science of measuring radiant energy transfers, which can be characterized using a set of physical quantities. These radiometric variables form a set of objective quantities.

Quantity	Symbol	Unit	Abbr.	Note
Radiant Energy	Q	Joule	J	Energy
Radiant Flux	Φ	Watt	W	Radiant energy per unit time, radiant power
Irradiance	E	Watt per square metre	W/m^2	Power incident on a surface
Radiance	L	Watt per steradian per square metre	$W/(sr \cdot m^2)$	Power per unit solid angle per unit projected source data
Radiant Intensity	I	Watt per steradian	W/sr	Power per unit solid angle

Table 2.1: Radiometric quantity, symbols and units.

Energy and Power (Flux)

Radiant Energy, Q, is the energy of a collection of photons. *Radiant power(flux)*, Φ, is the derivative of energy with respect to time. This expresses how much total energy flows to a surface per unit time.

Irradiance

Irradiance is the incident radiant power on a surface, per unit projected surface area.

$$E = \frac{d\Phi}{dA} \tag{2.2}$$

Radiance

Radiance, L, is the radiant flux per unit solid angle per unit projected area.

$$L = \frac{dE}{d\omega} = \frac{d^2\Phi}{d\omega dA \cos\theta} \tag{2.3}$$

Radiance is the most fundamental quantity in global illumination since it captures the appearance of objects in a scene. Radiance is invariant along straight paths; for any two mutually visible points x and y in space, the radiance leaving point x in the direction of point y is the same as the radiance on point y from the direction of point x. Radiance is represented in five dimensional space as L; for example, $L(x \rightarrow \Theta)$ is the radiance at x in the outgoing direction Θ, whereas $L(x \leftarrow \Psi)$ is the radiance at x in the incoming direction Ψ.

2.1.4 Material Properties

Materials in nature interact with light in many different ways. The reflecting properties of a material are explained by the concept of reflectance, which affects the appearance of objects.

BRDF

The most general expression for reflectance is the *bidirectional scattering surface reflectance distribution function (BSSRDF)* [NRH+77, JMLH01], which defines the relationship between the incident and reflected radiance. With the assumption that subsurface scattering is not considered, this equation can be

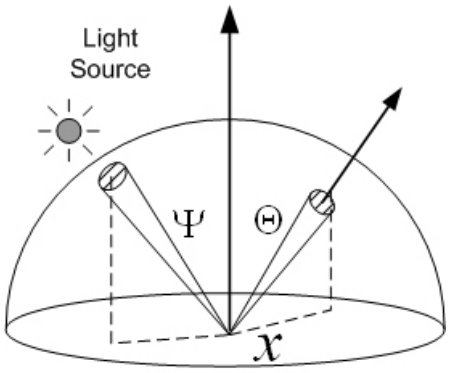


Figure 2.2: Bidirectional Reflectance Direction Function.

simplified to the *bidirectional reflectance distribution function*. The BRDF $f_r(x, \Psi \rightarrow \Theta)$ is defined as the ratio of the radiance in the outgoing direction Θ to the irradiance in the incident direction Ψ. It has a property called the *Helmhotz reciprocity*, which means that the change of incident and extant directions does not affect the amount of light that is reflected. By applying this reciprocity property, $f_r(x, \Psi \rightarrow \Theta)$ is equal to $f_r(x, \Theta \leftarrow \Psi)$ or simply $f_r(x, \Theta \leftrightarrow \Psi)$.

$$f_r(x, \Psi \rightarrow \Theta) = \frac{dL(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)} \tag{2.4}$$

$$= \frac{dL(x \rightarrow \Theta)}{L(x \leftarrow \Psi)\cos(N_x, \Psi)\, d\omega_\Psi} \tag{2.5}$$

This describes the directional distribution of reflected light. Two ideal cases are extensively used in computer graphics in order to simply these models, which are known as diffuse and specular reflectors. Diffuse surfaces reflect light uniformly in all directions. In the ideal diffuse case, the BRDF is an average all incoming radiances over hemisphere and redistribute equally toward outgoing directions, which is consistent with the law of reciprocity [SH92]. The perfect specular surfaces only reflect light only in mirror direction. The incident and exitant polar angle is equal, and the BRDF in this case is a Dirac distribution, $\delta(x)$.

2.2 Mathematical Foundation of Global Illumination

This section describes some mathematical foundations, such as the plenoptic function, the energy balance equation and Neumann Series Expansion. These describe the fundamental mathematical framework for solving global illumination problems, where the equations are solved in a numerical fashion in Section 2.2.3.

2.2.1 Plenoptic Function

The goal of global illumination in computer graphics is to compute a set of realistic radiances for a viewpoint. By employing the concept of the *Plenoptic Function*[AB91], the virtual world can be expressed

as a flow of light energy. The plenoptic function captures the flow of light in space, which means a set of rays visible from any point in space, at any time, and over any range of wavelengths can be determined as follows:

$$Plenoptic(x,y,z,\theta,\phi,\lambda,t) \tag{2.6}$$

The plenoptic function represents the radiances flowing through every position (x,y,z) in a scene in all possible directions (θ,ϕ) as illustrated in Figure 2.3. The parameter t describes the dynamic changes over time.

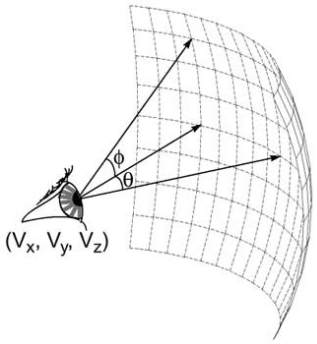


Figure 2.3: The Plenoptic function.

Ideally, the plenoptic function is continuous over the range of parameters. However, this can be generally simplified by ignoring time t and selective wavelength λ. Since the parameterization is in five dimensional space, it is seldom possible to compute and store the entire flow of light in a structure. However, an approximated solution to this is possible by limiting the range of input parameters. An attempt was made using the *light field* approach [LH96, GGSC96] to represent a 4D scalar function, which gives radiances at 3D spatial positions for every direction. However, this was limited to a bounded region of space and has a disparity problem (further explained in Section 2.4). In the following section, the mathematical foundation for equilibrium distribution of light energy in a scene is described as an integral equation.

2.2.2 Energy Balance Equation

The energy equilibrium of radiative surfaces is expressed by the following integral equation, which assumes the absence of participating media:

$$\underbrace{L(x\rightarrow\Theta)}_{\text{Total radiance}} = \underbrace{L_e(x\rightarrow\Theta)}_{\text{Emitted radiance}} + \underbrace{\int_{\Omega_x} L(x\leftarrow\Psi) f_r(x,\Psi\rightarrow\Theta)\cos(N_x,\Psi)d\omega_\Psi}_{\text{Reflected radiance}} \tag{2.7}$$

Kajiya [Kaj86] originally presented a slightly different form of the equation called the *Rendering Equation* in the context of computer graphics. In fact, this equation appears under various names, such as the *Energy balance equation, Radiance equation, Light transport equation, Global illumination equation, Scattering equation and Surface rendering equation*. This energy balance equation shows that the total outgoing energy at point 'x' is the sum of emitted radiance and reflected radiance. The first term on the right hand side is the emission. The second term on the right-hand side of the equation shows the effect

of light reflection as an integral over all of the possible irradiance energies ($L\cos\theta d\omega$) multiplied by the bi-directional reflectance distribution function.

The rendering equation is the mathematical foundation of the global illumination problem. This can express the exitant radiance $L(x \to \Theta)$ at any surface point x in any direction Φ. Therefore the rendering equation provides a solution to the plenoptic function in virtual environments.

2.2.3 Neumann Series Expansion

The rendering equation cannot be directly evaluated, since the radiance term is on both sides of the equation. It is convenient to use *Linear Transport Operator* notation to understand this equation in a compact form. By applying operator notation, the rendering equation can be written as:

$$L = L_e + T\,L \tag{2.8}$$

$$(I - T)\,L = L_e \tag{2.9}$$

The integral operator T describes the redistribution of radiance energy on the reflector from all surfaces of the scene. The definition of T is:

$$T = \int_{\Omega_x} f_r(x, \Psi \to \Theta)\cos(N_x, \Psi)d\omega_\Psi \tag{2.10}$$

By applying the inverse of $(I - T)$ to the emission function E, the rendering equation can be recursively evaluated as the *Neumann series* [Kaj86] expansion. If L_e is replaced by an emitter E E, then:

$$L = (I - T)^{-1}E \tag{2.11}$$

$$= (I + T + T^2 + T^3 + T^4 + ...)\,E \tag{2.12}$$

$$= \sum_{n=0}^{\infty} (T)^n E \tag{2.13}$$

The first of order of expansion ($L = I \cdot E$) describes emitters without any illumination. The second order ($L = (I + T)E$) provides direct illumination. This has the most effect on the solution; therefore it is very important to have a good approximation. Convergence of the Neumann series is guaranteed such that the amount of energy reflected from all surfaces is less than the incident amount of energy.

2.3 Overview of Global Illumination Methods

The physically-based simulation of light transport [Kaj86] in virtual environments is called global illumination. The goal of global illumination is to simulate all reflections of light, therefore reproducing an accurate intensity of the light at any given point. Global illumination simulates not only a path directly from light sources, but also represents indirect illumination, taking account of complex material properties and reflection models.

In this section the traditional global illumination techniques are examined, which primarily aim to deliver physical simulation of light transport. Many global illumination algorithms have been developed in the last few decades. Most of the early research was based on two major techniques, known as point sampled ray tracing and finite element radiosity. The rendering method developed in this research

group's study is very closely related to the techniques used in the ray tracing method, and shares the concept of point-based rendering that enables parallel computing on GPUs. The Monte Carlo ray tracing technique has also been used in our research to stochastically approximate the outgoing radiance values of the rendering equation. The rest of this section introduces hybrid methods and photon mapping. Hybrid methods combine both ray tracing and radiosity techniques, to take advantages of both algorithms. Photon mapping is a two-pass algorithm to solve the rendering equation using photons. Due to a lack of hardware development of graphics cards in the early stages of development, researchers mainly focused on the development of software for fundamental global illumination solutions, which involved exploration of some acceleration techniques. Early global illumination solutions were not capable of achieving interactive frame rates because of the high rendering cost of evaluating the rendering equation.

2.3.1 Finite Element Radiosity Methods

In the previous section, the algorithms that were developed to directly compute the intensity of light passing through a pixel in the image plane were discussed. In contrast, this section describes the methods that compute illumination in the object space. The finite element technique typically uses two step algorithms. Firstly, a scene is subdivided into small patches in order to compute radiances by solving a set of linear equations for light exchange between all the patches. The pre-computed values are stored in finite element data structures. In the next step, re-computed values are interpolated to generate the final resulting images. Introductions and overviews of the classic radiosity method can be found in textbooks [CW93b, SP94].

Radiosity [GTGB84] was developed for scenes with Lambertian surfaces, to compute inter-reflection efficiently using the *form factor* calculation. The fundamental operation in radiosity theory is the computation of these form factors, and Cohen et al [CG85] adapted the *hemicube* technique to reduce the computation time required to determine the visibility information. As an extension to form factor computation, ray-based techniques [WEH89, SP89] were employed to perform the numerical integration of the form factor equation. In further research, the radiosity solution was reformulated in a progressive refinement manner [CCWG88] so that the algorithm could produce the results to the complete radiosity equation progressively. This technique also allows calculation of the form factor on the fly. An extension to the progressive refinement solution was proposed in incremental radiosity [Che90] to offer an interactive modelling environment. The traditional separate processes were replaced with a single new approach, known as *rendering-while-modelling* as described in the research paper.

The computational complexity of the radiosity solution has been addressed in early development. In comparison with ray tracing methods, the high cost for complex models comes from the fact that the radiosity algorithm computes values for every patch in the model. Many techniques have been developed to improve the efficiency of the algorithm. The radiosity method has also been extended to the importance-driven approach [SAS92], the clustering algorithm for complex environments [SAG94] and the hierarchical radiosity algorithm [HSA91]. These techniques reduce the time and complexity of the radiosity algorithm with a subsequent reduction in the accuracy of the solution.

With increasing the number of patches in a scene, the form factor calculation becomes the major

bottleneck in radiosity solutions. A number of algorithms use form factor sampling based on uniformly distributed global lines. For instance, the global lines are generated by connecting two sample points on the bounding sphere for the scene. This algorithm has been used in many radiosity applications [Sbe93, Pel95, Sbe97, KP98]. In other studies, there have been attempts to make use of wavelet theory in radiosity solutions [GSCH93, CSS96], and further to this, radiosity was extended to handle non-diffuse environments [LTG92]. Finally, in this thesis, hardware acceleration techniques are discussed in Section 2.6.1

2.3.2 Ray Tracing Methods

The fundamental idea of Ray (or Path) tracing is to generate light transport paths between light sources and the point in the virtual scene, in order to compute radiance values. These methods usually calculate radiance values for each pixel in the screen space directly. For this reason, this category of techniques belongs to pixel-driven (or point-sampled) ray tracing algorithms. A general overview of ray tracing algorithms is discussed in [Gla89]. The ray tracing algorithm for global illumination was initially introduced by Whitted [Whi80]. This method traces rays in a backward path from the observer to the light source, to simulate perfect specular reflection, refraction and direct illumination with visibility determination.

Acceleration Techniques and Further Optimization

Since the introduction of Kajiya's rendering equation, the global illumination community has tried to develop algorithms that can render realistic images to some accuracy in a reasonable amount of time. A practical scene using Monte Carlo ray tracing requires generation of a large number of rays per pixel. Reducing the number of primary or secondary rays can lead to significant improvements in rendering time.

The number of secondary rays per pixel can be large for a scene with many light emitting and specular objects. For shadow rays, it is sufficient to find any occluding object to guarantee occlusion. For this reason, Haines [HG86] proposed a *light buffer* for shadow caching, exploiting the coherence stored at each light source. This idea has been extended to a sophisticated optimization for shadow rays [FBG02]. This algorithm subdivides the scene into a set of voxels, which stores a list of light sources that are occluded, visible and partially visible. Another algorithm for reducing the number of shadow rays was introduced in [War91], where light samples are sorted according to their contribution.

Apart from reducing the number of rays, another obvious approach for acceleration would be an improvement of the core ray tracing algorithms, such that each ray can be traced faster. As Whitted [Whi80] explained, most of computation time in ray tracing is spent in ray primitive intersections. A large number of different algorithms were dedicated to increasing the speed of ray primitive intersections [MT97, Bad90, Woo90]. However, it is still costly to intersect a ray to all primitives in complex objects. To avoid this, a tightly enclosing *bounding volume*, which is normally a simple geometric primitive, was introduced. When a ray misses the bounding box, the complex object can be disregarded cheaply. The most efficient way to accelerate ray tracing is to exploit spatial and hierarchical data structures to reduce the number of ray primitive intersections per pixel. Many different kinds of acceleration structures

were developed in recent decades, such as uniform [AW87, Coh94], non-unform [KS97], recursive grid [JW89], hierarchical grid [CDP95], Octrees [Gla84, WSC+95], Bounding Volume Hierarchies [KK86, Smi98], BSP [SS92], Kd-tree [Hav01, Bit99] and higher-dimensional ray classification [AK87, LW95]. Among all of these techniques, hierarchical subdivision methods reduce the computational complexity most effectively, from $O(N)$ to $O(\log N)$.

The ray tracing algorithm has been optimized to run at real-time rates. Parker et al [PMS+99] demonstrated that a ray tracer can achieve an interactive rate on large, shared-memory supercomputers. They have proved that ray tracing scales well in multiple processor environments. Pharr et al [PKGH97] exploited coherence by reordering the tracing computation, thereby achieving interactive frame rates. Wald et al [WBWS01, Wal04] optimized the implementation of the ray tracer method using caching and SIMD optimization on a CPU for complex environments. The most recent techniques to accelerate ray tracing are based on exploiting graphics hardware as parallel streaming processors, which will be discussed in Section 2.6.2

2.3.3 Monte Carlo Path Tracing Methods

Cook et al [CPC84] presented a distributed ray tracing technique where rays are distributed stochastically for all the light paths, to simulate fuzzy phenomena such as motion blur, depth of field, penumbras and glossy reflections. Cook [Coo86] also presented a non-uniform sampling scheme to perform a Monte Carlo evaluation of integrals in a stochastic manner, in order to reduce aliasing artifacts presented in point sampling schemes. Kajiya [Kaj86] generalised Monte Carlo path tracing to the rendering equation to simulate all types of optical phenomena, including caustics and inter-reflections between any types of surface. Arvo et al [AK90] adapted some statistical techniques to stochastic path tracing, one of them being *Russian roulette*, which can terminate the recursive tracing in an unbiased way. The most complete solution is bidirectional path tracing [LW93, VG94], where rays are traced simultaneously from the light and the eye. However, these pure unbiased Monte Carlo based ray-tracing methods are still very time intensive. In contrast, our proposed solution uses the Monte Carlo path-tracing method on the GPU to simulate the rendering equation in a stochastic manner. The GPU-based approach allows a geometry acceleration structure to be built very efficiently, so that real-time rendering of path-tracing can be achieved.

The most common problem that occurs with stochastic models is variance (or noise) in the rendered image. A straightforward solution to this problem is to have a large number of sample rays. Many researchers have proposed algorithms to reduce the noise by carefully distributing rays, such as Basic lighting condition [Shi91], sampling techniques [VG95] and Metropolis Light Transport [VG97].

2.3.4 Hybrid Methods

In the previous sections, an evolution of two major algorithms were described, where the fundamental concept of these two methods remained the same. A ray tracing algorithm computes radiance values for each pixel by finding paths between the pixel and the light sources, whereas a radiosity solution computes radiance values for every element in the scene. In general, radiosity can simulate diffuse reflections well, and ray tracing is good for specular reflection models. It seems natural to combine the two to obtain the

advantages from both schemes.

The first hybrid techniques [WCG87, SP89] used the radiosity method to compute diffuse reflection and used ray tracing to handle mirror effects. Shirley [Shi90] proposed a three pass rendering method, where the pre-computed solution is used only for indirect lighting. Many researchers [CRMT91, ZS95] noticed visible artifacts in the radiosity algorithm, such that path tracing was employed to account for all types of light scattering directly. These algorithms only used the radiosity method to compute the indirect illumination of the diffuse surfaces, and used Monte Carlo ray tracing for all the details required in the final image.

Particle Tracing is a method used in the application of Monte Carlo techniques to simulate lighting travelling from the light source to the scene using stochastic methods. The difference compared to path tracing is that the particles are considered to carry energy. Bidirectional path tracing [LW93, VG94] generates paths starting at the light source and at the surface point, where both paths are connected to determine the energy contribution. It has the advantages of both ray tracing and particle tracing.

Since the Monte Carlo rendering takes a long time to generate adequate quality images, the idea of reusing radiance or irradiance values has been developed. Ward et al [WRC88] introduced an irradiance caching scheme to accelerate the computation of indirect illumination. This approach is based on the fact that the irradiance at diffuse surfaces varies smoothly. *Irradiance Volume* [GSHG98] is extended to five dimensional space such that irradiance values are stored in a regular grid structure to account for all points and directions. Ward and Heckbert [WH92] developed *Irradiance gradients* to determine when the cached values can be interpolated to produce reasonably accurate results. Other suggested caching schemes are Render Cache [WDP99] and Octree based caching algorithms [WS99].

2.3.5 Photon Mapping (Multi-Pass) Methods

Photon mapping [Jen95, Jen96] is a robust two-pass algorithm that traces illumination paths both from the light and from the viewpoint. In the first pass, a *photon map* is constructed from photons that are emitted from the light sources and interact with all of the surface types in the scene. In the second pass, the values stored in the photon map are gathered to formulate final rendering images. Unlike other bidirectional path tracing methods [LW93, VG94], this technique caches and reuses photon values that are stored in separate independent data structures. The caustics map is specifically designed to store the photons that interact with specular surfaces. The combination of photon mapping and a Monte Carlo ray tracing based rendering algorithm results in being significantly more efficient than conventional Monte Carlo ray tracing. Several extensions have been added to photon mapping [PP98, SW00, Jen97, Chr99, PDC$^+$03, CB04]. For example, a significant addition is light interaction with participating medium [JC98].

2.4 Precomputed Methods for Global Illumination

The plenoptic function captures a flow of light, which is also in the image-based rendering domain. The first attempt at this was made in the *light field rendering* approach, which is described in detail in this section. The first step is to introduce the path that image-based rendering has developed, to use radiance

images (or photos) to reconstruct a new scene at any viewpoint. Then some parameterizations will be described to explain the *light field* in a uniform manner. The acceleration structure developed in this thesis by our research group stems from the fundamental idea of the *light field* model, where a function of radiances is associated with the surface geometry. However, the new approach differs significantly, since it stores depth information to build a 5 dimensional (positional and directional) visibility structure using hardware acceleration in order to efficiently compute light transport for multiple objects in the virtual environment.

A new notion of rendering techniques using a set of images has emerged, which offers an alternative solution to traditional geometric methods for generating more realistic results from complex environments. Image-based rendering is a technique that allows interaction with objects and scenes with original specifications from digitized photographs or from synthesized images. A set of input images is used to build new high quality images of scenes or objects with low computational costs.

2.4.1 Image-Based Rendering

Image-based rendering covers a wide range of different techniques, all of which use images as a significant component. The most commonly used image-based approach is texture mapping [BN76], where the appearance and complexity of an object is represented by an image. Environmental mapping [Gre86] has been a popular technique for image-based rendering. Multiple environment maps are created from cylindrical panoramic images at discrete points, which are used to compose images seen from locations with continuously changing view directions. Seitz et al [SD95, SD96] proposed image morphing, which generates a series of intermediate images between two or more reference images using basic principles of projective geometry.

Apple's QuickTime VR [Che95] generates a single panoramic image to navigate the virtual environment. This rendering method allows for rotations and modifications of the field of view within the three dimensional environment. Although dynamic perspective changes in real-time are possible, this is limited to a single viewing position. Some other studies have concentrated on a rendering technique using view interpolation [CW93a]. A hybrid approach [DTM96] has also been proposed, which combines both geometry and image based techniques for modelling and rendering architectural scenes from a set of synthetic images. All of these methods fall into the category of modelling and rendering using image warping and interpolation.

2.4.2 Light Fields (Lumigraph)

Early research showed that image-based rendering techniques can offer an alternative solution to traditional geometric methods for modelling and rendering complex objects. An interesting concept in image-based rendering is to consider a collection of images as a database of rays with no associated structure. New images are approximated from these collections by interpolating between nearby rays, which allows capturing of the complete object appearance directly from real world image data, without building any geometry.

Levoy and Hanrahan [LH96] proposed a new paradigm approach in computer graphics in which the underlying modelling primitives are considered as rays rather than images, in contrast to the previous

image-based rendering methods. Light field rendering [LH96] represents the flow of light in radiance rays. This idea is based on the plenoptic function [AB91], which captures the complete flow of light in a region or environment. The study suggested two-plane parameterization, where the set of lines in space is parameterized by the intersection points of each line with two planes, in order to capture all of the rays as they pass through a slab of empty space. Gortler et al [GGSC96] propose a single arrangement of six pairs of planes called the Lumigraph. The Lumigraph represents all the light as a 4D function in unobstructed space, where the camera is positioned outside of the convex hull object.

The light field rendering method has several advantages. It is capable of generating global illumination with a combination of specular and diffuse materials and caustics. In addition to this, the light field obtained from both synthetic and real images can be rendered in real-time, independent of the complexity of the scene. However, this relies on either a very dense set of images, or sophisticated reconstruction algorithms in order to synthesize new images. In practice, the results suffer from ghosting problems, limited fields of view and costly storage requirements. The storage requirement for a light field is very large, so that a lossy compression technique is used to minimize some of the redundancy in the light-field representation. Other issues were discussed and resolved in [SCG97, IMG00, HLCS99]. The parameterization used in the light field approaches shows noticeable artifacts when the camera crosses the boundary between two light slabs. Even doubling the number of light slabs is not sufficient to avoid the disparity problem. A solution to this requires a uniform representation that is invariant under both rotation and translation. To this end, a few techniques for capturing the light field in a uniform fashion have been proposed [CLF98, CF99, IPL97].

Uniformly Sampled Light Fields

Camahort et al [CLF98, CF99] suggested two new uniform representations for light field modelling. The first is a two-sphere parameterization, which uniformly subdivides a sphere into hierarchical elements. The light field is sampled by joining pairs of subdivision elements in a multi-resolution fashion. The second method, known as sphere-plane parameterization, allows a uniform sampling of all five dimensions of the light field, using hierarchical subdivision for directional space, and uniform grid sampling for positional space. Light field models are acquired using parallel projections along a set of uniform directions. Ihm et al [IPL97] also proposed a spherical light field, which constructs the plenoptic function as a collection of small, uniformly subdivided directional spheres that cling to a large positional sphere. In this manner, the complete flow of light can be parameterized with four parameters in spherical coordinates, where each point on the surface of the directional sphere is parameterized by two variables. An oriented ray is determined by associating a direction with each point on the positional sphere. The use of the sphere provides a symmetric representation of the complete flow of light. The idea of exploiting geometry information for parameterizing the light field was proposed in the Surface Light Field method by Wood et al [WAA$^+$00]. This is a function from the surface to a *lumisphere*, where a lumisphere is a set of directions of radiance color, which represents the radiance leaving a point in all directions. This study used the generalization of Vector Quantization and Principal Component Analysis to compress light fields at interactive rates. This idea was extended to light field mapping [CBCG02] to enable

the use of the surface light field in real-time rendering, by developing a compact representation for the graphics pipeline.

Our parameterization of the SLHB structure is similar to that proposed by Camahort et al [CLF98, CF99], where a light field is parameterized in a uniformly sampled way. The new method put forward in this thesis also uses uniformly subdivided spherical directions to build an LHB for each direction.

2.4.3 Pre-computed Radiance Transfer

The light field approach captures the flow of light in 4D functions in the pre-processing stage, and synthesizes the final image from the obtained images. Another major development in the pre-computed method is the Pre-computed Radiance Transfer (PRT) technique proposed by Sloan et al [SKS02]. This method offers real-time rendering with complex lighting interactions, by saving pre-computed illumination data in the pre-processing step. Spherical harmonics are used to represent the radiance transfer function between surfaces of an object. PRT encodes illumination information in a compact form as spherical harmonic coefficients. The pre-computed transfer information is then applied to incident lighting to compute the global illumination solution in real-time, using graphics hardware. Although this is applicable to real-time global lighting effects, it requires an expensive pre-computation step, which is incapable of applying the technique to real-time dynamic environments. However, our solution does not require any pre-processing, but can handle all aspects of dynamic elements on the fly. Although many high-order spherical harmonics can express some fine detail, PRT solutions are mainly limited to low-frequency lighting environments in order to handle a reasonable number of spherical harmonic coefficients. This limitation can be overcome by replacing spherical harmonics with discrete wavelet representation [NRH03]. Liu et al [LSSS04] proposed another way of replacing these spherical harmonics, by adapting a clustered principal component analysis as a quantization method. However, their PRT method can handle a glossy object at interactive rates for a limited 50K polygons, whereas our solution can handle over one million polygons at interactive rates.

2.5 Interactive Global Illumination

Real-time global illumination has been a very active research area in the past decade. This section will give a brief overview on the early development of interactive global illumination solutions. Despite significant advances in graphics hardware in recent years, it is still challenging to achieve physically correct indirect illumination in real-time without pre-computation. The previous section shows that pre-computed solutions can offer real-time rendering for global illumination, but only with a high cost associated with pre-processing. Several global illumination algorithms will now be presented, which offer interactive frame rates for mainly diffuse environments. Many of the global illumination methods introduced in Section 2.3 and 2.4 compute direct and indirect illumination as accurately as possible by intersecting the rays with the geometry. The following methods allow the rendering of global illumination at interactive frame rates with the loss of some accuracy. Interactive illumination for dynamic environments can be easily achieved if visibility for indirect illumination is completely neglected [DS06]. Arikan et al [AFO05] approximated a global illumination solution where the nearby geometry is integrated without

computing the visibility all. The fundamental idea of improving the rendering performance is to speed up the indirect illumination efficiently by using an alternative approximated solution.

2.5.1 Instant Radiosity

Instant Radiosity (IR) was first introduced by Keller [Kel97], who suggested the new idea of *Virtual Point Light*. IR is a simple global illumination algorithm that approximates diffuse lighting of a scene by placing virtual point lights, which act as indirect light sources. This technique has advanced a new level of interactive global illumination solutions for diffuse environments. The algorithm is fairly straightforward; for each light source, some photons are sent into a scene. At each randomly chosen intersection point, a VPL is generated to act as a new indirect light source. The final image is approximated from direct and indirect lighting, with virtual point lights acting as each light source. The advantage of the IR solution is that complex indirect illumination is computed from only a handful of the set of virtual point lights. As this method does not require any pre-computations of the scene, it supports interactive movements of dynamic objects. There are many follow up studies to this [CPWAP08, NW09, WWZ+09] that were intended to improve the original idea. The lightcuts method [WFA+05] groups a large number of VPLs into a hierarchy to speed up the rendering. Laine el at [LSK+07] improved instant radiosity by adapting their caching scheme. Over several frames, their technique checks whether buffer [Kel98] to reduce the number of samples per pixel. This technique is applied in the gathering of indirect illumination data only. It is important to note that early developments of IR were bounded to mostly static environments and hard-to-handle dynamic objects.

2.5.2 Imperfect Shadow Map

Although instant radiosity [Kel97] and instant global illumination [WBS03] simplify the rendering equation by using virtual point lights, both methods still require accurate visibility to compute indirect illumination. The fundamental bottleneck of real-time graphics in global illumination is the high cost of this accurate visibility test for indirect lighting computation. Ritschel et al [RGK+08] suggested an *Imperfect Shadow Map(ISM)* to overcome this bottleneck, by adapting point representation. The basic idea is to use an incorrect or rough estimation of the visibility information to speed up the indirect lighting computation. In order to do this, a scene is represented as a large collection of point clouds, so that a fast shadow map can be created from the point cloud for every VPL. In this way, hundreds of shadow maps can be generated per frame to allow dynamic movements. Although ISM can achieve interactive frame rates for dynamic scenes with reasonable complexity, it fails to cope with highly complex scenes due to inefficient management of point representation. ISM methods tend to wipe out the detail of indirect shadow due to an overly simplified representation. In contrast, the method presented in this author's thesis preserves the detail by using the accuracy model geometry, rather than point representation.

Ritschel [REG+09] also extended the point representation to hierarchical representation of splats in order to handle complex scenes. This method uses data gathering based on a CUDA architecture, which is an SKD that enables graphics cards to be used for various purposes. Our rendering algorithm is also built on CUDA and OpenGL architecture, and it also shares the fundamental idea of using approximated visibility in the rendering method. However, our solution provides simulation for fully dynamic scenes,

with multiple bounces of indirect illumination at real-time frame rates for moderately complex scenes, and interactive rates for highly detailed objects consisting of over one million polygons.

2.6 Overview of Global Illumination on GPU

In this section, early GPU methods for global illumination are introduced. The most computationally expensive part of the rendering equation lies in the visibility calculation, and our solution is not an exception to this. Many researchers have proposed ways to accelerate the visibility information using graphics hardware in global illumination problems. In this thesis, an acceleration structure for fast visibility query is presented in order to achieve this. It employs several GPU techniques to accelerate visibility and to enhance propagation timing in the computation of the rendering equation. More details of this are presented in chapter 3.

The development of the GPU (Graphics Processing Unit) on consumer graphics hardware has resulted in significant improvements in processing power, memory and programmability. A parallel architecture of the GPU has enabled the performance to develop faster than the CPU capability. For this reason, the GPU is used in not only the computer graphics field, but also for general parallel problem solving. An introductory survey of general purpose computation using the GPU was presented in [OLG+05, LHK+04].

The functionality and programmability of the GPU was limited until the OpenGL 2.0 extension was introduced. This allows the user to program parallel problems directly on graphics cards with the precision of 32-bits floating point, which results in numerical computation being more accurate. A new feature, known as the Frame Buffer Object (FBO), enables rendering of a texture to memory without a readback from the main memory, which was the major bottleneck of the previous generation. In addition, the advanced compiler Nvidia's CG [MGAK03] and Microsoft's HLSL [KBR04] have reduced the programming time by offering high-level language capability on the GPU. There will now be a discussion of the approaches that employ the GPU in the global illumination domain.

2.6.1 GPU Radiosity Methods

Radiosity methods are an effective solution to the global illumination problem in diffuse environments. Radiosity requires a computationally expensive preprocessing step in order to form the factors computation from every visible patch. However, once the radiosity is obtained, a real-time walkthrough of the virtual environment is possible. Typically, the preprocessing phase is performed off-line, and many researchers have attempted to exploit graphics hardware in the GPU,

Nielsen et al [NC02] partially exploited graphics hardware to compute the hemicube [CG85] for visibility computation using texture mapping. Car et al [CHH03] used the floating point textures, applying a Jacobi iteration on graphics hardware to find the matrix radiosity solution. The algorithm can support dynamic relighting assuming the geometry is fixed. Coombe et al [CHL04] presented a technique that requires no preprocessing, which performs the entire computation, including form factors, visibility and shooter selection implemented on the GPU. A progressive refinement radiosity solution [CCWG88] is chosen, since radiosity values can be stored in hardware texture memory, which can be

used for rendering. They also implemented an adaptive subdivision method on the GPU to reduce the visibility computation.

2.6.2 GPU Ray Tracing Methods

Ray tracing [Whi80] is one of the classical solutions in computer graphics. Numerous global illumination methods are based on ray tracing techniques, including path tracing [Kaj86], Monte Carlo ray tracing [CPC84, Coo86, AK90] and photon mapping [Jen96]. For this reason, the graphics community has been eager to improve the speed of ray tracing, using whatever resources are available [Bad90]. A coherent ray tracing method that optimizes SIMD instructions was presented by Wald et al [WBWS01], and for larger scenes on a shared memory multiprocessor and a cluster [RCJ98]. Special purpose hardware also supports efficient parallel ray tracing [WSS05]. One of the most time consuming operations in ray tracing is to compute the visibility information in object space, which is a ray-object intersection problem.

The Ray Engine [CHH02] is one such application, which configures the GPU to compute ray-triangle intersections. Due to frequent communication between the CPU and GPU, there is a subsequent degradation in performance. The ray engine could be used to accelerate path tracing, Monte Carlo ray tracing, photon mapping, form factor computation and visibility pre-processing. Purcell et al [PBMH02] also presented a similar study on real-time ray tracing, such that the GPU is employed as a streaming processing model for a highly parallelized algorithm. Their work has led to several other GPU ray tracing implementations [Chr05, KL04]. Early ray tracing models [CHH02, PBMH02] were incapable of utilizing spatial coherence. Instead of a simple uniform grid, Simonsen et al [ST05] applied a bounding box hierarchy to the GPU as an an acceleration structure. A Kd-tree structure on the GPU is implemented by Foley et al [FS05], which is several times faster than a uniform grid. Carr et al [CHCH06] presented an efficient solution, which uses a threaded bounding volume hierarchy stored as a geometry image MIP map, in order to quickly intersect triangular meshes on the GPU. This method can update the hierarchy in every frame such that the ray can trace dynamic geometry.

2.6.3 GPU Photon Mapping Methods

The ray tracing method presented in [PBMH02] has been extended to photon mapping in [Pur04, PDC+03] by resolving the sorting and searching problem, which commonly appears in streaming frameworks. Another study in photon map rendering was presented in [MM02]. This method tries to resolve the kd-tree searching problem in the hardware by adapting a hashing algorithm to find the nearest neighbors. Since the GPU architecture is not capable of handling a complex kd-tree structure, both of the approaches [PDC+03, MM02] used simplified structures neglecting the performance. Larsen et al [LC04] proposed a combined approach by balancing loads between the CPU and GPU. The GPU accelerates final gathering and caustic filtering, whereas the CPU traces caustic photons in a data structure.

An approximated solution to caustic rendering in real-time has been suggested in [WS03]. This algorithm chooses sample points on the specular surfaces, where each sample point functions as a pinhole camera that projects incoming light on diffuse receivers. Although it can render caustics in dynamic scenes in real-time, it is limited to a single specular interaction.

2.6.4 GPU Irradiance and Radiance Caching Methods

Ward et al [GSHG98] introduced an irradiance caching scheme to take advantage of the fact that irradiance has a low variance. Irradiance is stored in a sparse data structure and used in gathering by interpolation. Nijasure et al [NPG03, NPG05] have extended the irradiance caching method to non-diffuse global illumination computation using graphics hardware. A set of regular sample points are chosen to store the incoming radiance function as spherical harmonics coefficients. Then irradiance at any surface point is approximated by interpolating the nearest sample locations. A drawback of this method is that the sample points are located independently of the light condition and complexity. Gautron et al [GKBP05] reformulated the irradiance and radiance caching algorithms [GKPB04, KGBP05] to allow GPU implementation. Their method has improved performance over classical irradiance caching by adapting a splatting scheme and radiance gradient computation [KGBP05] instead of conventional nearest neighbor interpolation.

(a) Accurate Area Light (b) Ambient Occlusion

Figure 2.4: Comparison of Accurate Lighting and Ambient Occlusion.

2.7 Screen Space Methods for Real-Time Global Illumination

In recent years, it has become popular to approximate global illumination based on screen space information. By limiting the computation space, this often produces inaccurate or fake GI, but in real-time rendering. Most computation takes place in the screen space, which is easy to implement in the GPU. Therefore, algorithms in this category often deliver high frame rates with minimum rendering cost. Our proposed rendering algorithm is based on a global interaction rather than a screen space in order to fully simulate all inter-reflection effects. However, it is possible to limit the visibility data structure to a single direction, which is a camera view. In this way, the building performance of the SLHB acceleration structure can be significantly increased, trading off some accuracy to improve the speed. Nichols et al [NPW10] presented a similar concept based on a layered depth image in screen space to compute

gathering illumination. Using a single layered hit buffer, the screen space path tracing method can be approximated; however, it cannot handle the situation where reflectors are outside of the screen space.

2.7.1 Screen Space Ambient/Directional Occlusion

Ambient Occlusion (AO) [ZIK98] is a technique that approximates the amount of light reaching a point over the hemisphere. A single constant value can represent an integral of the occlusion contribution on that point. AO algorithms can add realism to a scene by using a pre-computed occlusion value to generate smooth soft shadows. Bunnel [BUN05] used an ambient occlusion method to compute the indirect illumination efficiently. The visibility between elements in a hierarchical link structure is estimated by a simple AO.

Screen Space Ambient Occlusion (SSAO) uses an image space to compute ambient occlusion. The depth information in the frame buffer gives an approximation of the scene structure. This is a similar concept that is used in our research, in order to build a visibility structure. In contrast, here we employ an orthogonal parallel projection to construct depth buffers instead of perspective viewing. Also, a multiple layered depth information is stored in the buffers along with polygon ID, so that a scene structure can be estimated from neighbor depth information. SSAO techniques are widely used in the games industry as an alternative solution to global illumination, since they are easy to implement in the GPU with a very small requirement of computational resources. Ritschel et al [RGS09] presented a Screen Space Directional Occlusion(SSDO) which is similar to the SSAO method. This adds directional visibility and indirect illumination from near by screen space geometry. This only adds minor computational time to SSAO by sampling the neighboring screen space pixels, which are then used to calculate indirect illumination.

2.7.2 Reflective Shadow Maps

Dachsbacher et al [DS05] introduced a screen space based technique, called *Reflected Shadow Map(RSM)*. This is an extension of the instant radiosity technique, providing a way to quickly generate high quality VPLs. A standard shadow map is extended to support the reflective shadow map, where every pixel is considered as a light source. By efficiently generating VPLs on the shadow map, RSM is able to interpolate indirect lighting in the screen space, yielding interactive rates for complex scenes. The solution is mainly limited to single bound indirect illumination. One of the drawbacks of this solution is that indirect illumination does not take into account the visibility calculation, because the cost of generating shadow amps for each VPL is too high. In this sense, our proposed visibility structure (SLHB) would be an ideal solution to provide approximated visibility without building many shadow maps. By adapting SLHB, the instant radiosity solution can benefit by computing efficient indirect illumination for real-time rendering of dynamic scenes. The key idea behind this new method is to use accurate geometry with an approximated visibility structure. In other words, the subset of all possible visibility fields is computed into the spherical layered buffer structure, which is used for visibility query in indirect lighting computation, which saves considerable time. By using screen-space interpolation of the indirect lighting, interactive rates can be achieved, even for complex scenes.

2.8 Virtual Light Field Method for Global Illumination

In this section, the early works of our research group will be introduced, such as the Virtual Light Field (VLF) approach [SMKY04], and its extension exploiting the GPU to speed up the propagation [KSMY07, MYK+08]. The early works inspired the new acceleration data structure presented in this thesis. The VLF is an algorithm that provides real-time walkthrough for globally illuminated scenes, with mixtures of ideal diffuse and specular surfaces.

A 2D grid of rays parallel to the z-axis is called the canonical parallel subfield (PSF). This canonical PSF is intersected with objects in the scene. It requires only a 2D rasterization algorithm to build the PSF structure. In order to ensure every polygon is added and propagated in the discrete grid representation, a continuous clipping is adapted, which takes most of the computing time. Multiple rotations of the PSF form a representation of the scene structure in a set of directional global rays. This structure inspired our proposed solution, which uses an acceleration structure in Spherical Layered Hit Buffers (SLHB). The VLF approach uses PSF to store computed radiances and also propagate them in a pre-processing stage, whereas our SLHB stores only depth and polygonal information. The idea of both data structures stems from Layered Depth Images [SGHS98], where each ray maintains a list of radiance or depth information at all intersection positions, so that a projected image can be reconstructed from any viewpoint and direction in the scene.

The VLF is based on the concept of Light Field methods, where energy propagation is done in a pre-processing stage, which stores the radiances in the light field structure for final image rendering. One of the major drawbacks of the VLF algorithm is a long propagation time and high memory requirements. However, it offers a global illumination solution for real-time walkthrough. It mainly relies on fast lookup at the final rendering stage. In other words, the final image synthesis is independent of scene complexity. A typical small scene requires at least one gigabyte of memory to propagate the scene. The vast amount of time in propagation is allocated to the clipping process, and [MYK+08] overcame this problem by parallelizing the clipping process using GPU processors. Although this GPU-based VLF approach enhances propagation and rendering speed, the algorithm can only handle static environments and still sacrifices memory and propagation time for interaction. In contrast, our proposed solution in this thesis is fully capable of running dynamic environments.

The VLF approach has also been integrated with the VR system for real-time GPU-based rendering in the Cave Automatic Virtual Environment [MKYS07a, MYK+08, KYMS06]. By separating the scene geometry into static and dynamic elements, it was possible to pre-compute the mainly diffuse global illumination for the static elements using the VLF approach, and some simple techniques to deal with dynamic elements, such as avatars and shadows. This was in order to make them realistic by synthesizing both elements. Some user studies were also conducted using VLF approaches to understand whether visual realism induces greater presence in immersive virtual environments, and our recent papers [SKMY09] and [YMKS11] indicate that visual realism enhances a realistic response in an immersive virtual environment. There have also been some other attempts to employ the VLF technique in ray-tracing solutions, such as [KMYS04, MKYS07b].

2.9 Summary

In this chapter, an overview of global illumination algorithms has been presented, which are capable of interactive or real-time rendering of virtual environments. Some algorithms use approximated global illumination techniques, such as virtual point lights or screen space computation to provide high frame rates. Recent development of customizable graphics hardware opened a new research area for real-time global illuminations. Our new algorithm has also benefitted from massive parallel processing power using a GPU. This solution is related to many ideas discussed in this chapter, where an interleaved sampling scheme and a discontinuity buffer are used in our algorithm to reduce the number of sampling directions. Our fundamental visibility data structure stems from the LDI [SGHS98] structure, which combines the multiple radiance and depth values representing the many polygons seen along a ray. Instead of perspective camera view projection to build multi depth maps, an orthogonal projection is used to build multi layered hit buffers in spherical directions, which contain 3D visibility data structures. In this way, approximated visibility structures are used for fast indirect illumination computation. Our acceleration structure is also related to a concept used in light field approaches [LH96, GGSC96, WAA$^+$00]. Light field stores radiance information in 5D structures, whereas our method stores depth values and associated polygon IDs. Finally, a stochastic ray tracing method has been employed [CPC84, Coo86, AK90] to solve the light transport problem [Kaj86].

Chapter 3

Global Illumination with Spherical Layered Hit Buffers

3.1 Overview

The goal of global illumination is to compute the converged distribution of light energy in a scene. To compute this distribution, an understanding of mathematical formulation is required, in order to describe global illumination in a numerical way. The following section will give an overview of mathematical approaches used to evaluate the rendering equation in terms of two operators; *Hemisphere Formulation* and *Area Formulation*. Then a novel acceleration structure is introduced, using Spherical Layered Hit Buffers (SLHB), which gives the approximated visibility field of the scene. The data structure holds a complete list of all intersections and polygon IDs. Some techniques are introduced to generate random directions over the (hemi)sphere in a uniform manner. Finally, there is an analysis of spherical directional distribution in terms of solid angles.

3.2 Mathematical Formulation of Global Illumination

Global illumination solutions aim to compute a physically accurate estimate of the function $L(x \rightarrow \Theta)$, which is the radiance at point x in three-dimensional space, in the direction Θ. Kajiya [Kaj86] showed that this function can be evaluated as an integral equation. In this section, the mathematical background of global illumination and the rendering equation (Equation 2.7) will be described. Two numerical light transport operators are presented to approximate the integral equation, and the rendering equation is iteratively solved using Neumann series expansion. Note that the notation used in this thesis follows Dutré's notation methods [Dut96].

Following is the common notation used to describe the flow of light:

- $L(x \rightarrow \Theta)$: radiance leaving point x in direction Θ
- $L(x \leftarrow \Theta)$: radiance arriving at point x from direction Θ
- $L(x \rightarrow y)$: radiance leaving point x, arriving at point y
- $L(x \leftarrow y)$: radiance arriving at point x, coming from point y

3.2.1 Hemisphere and Area Formulations

The Rendering Equation formulates the equilibrium distribution of light energy in a scene. This equation computes the outgoing radiance $L(x \to \Theta)$ in direction Θ at a surface point x. Light propagates instantaneously in non-participating media space. There are two commonly used methods for solving the rendering equation, which are *Hemisphere integration* and *Area (Surface) integration* methods as illustrated in Figure 3.1.

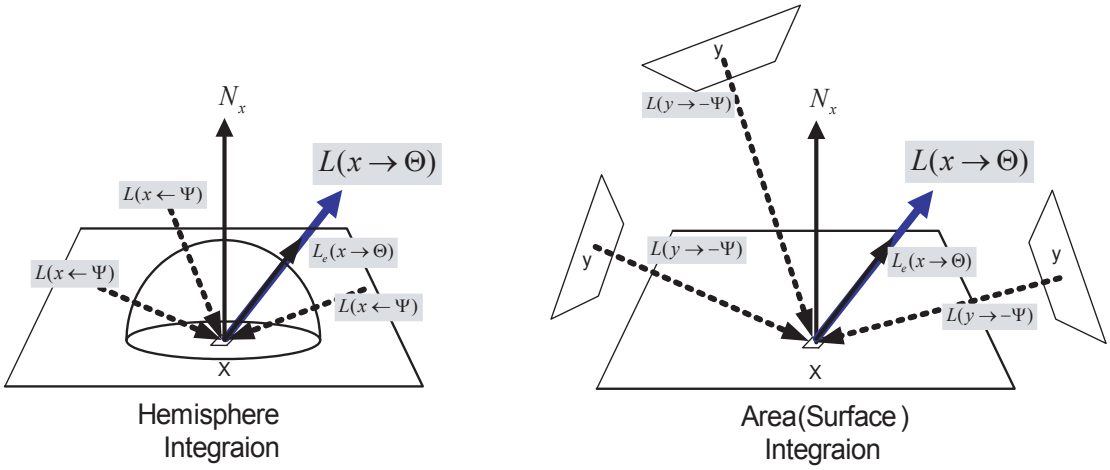


Figure 3.1: Hemisphere and Area (Surface) Integration.

Hemisphere Formulation

One of the most widely used formulations of the rendering equation is the hemisphere formulation, where the rendering equation is derived using energy conservation theory. The total outgoing radiance $L(x \to \Theta)$ at a point x in a particular direction Θ is the sum of the emitted radiance $L_e(x \to \Theta)$ and the reflected radiance $L_r(x \to \Theta)$:

$$L(x \to \Theta) = L_e(x \to \Theta) + L_r(x \to \Theta) \tag{3.1}$$

Recall the BRDF function definition:

$$f_r(x, \Psi \to \Theta) = \frac{dL_r(x \to \Theta)}{dE(x \leftarrow \Psi)} \tag{3.2}$$

$$dL(x \to \Theta) = f_r(x, \Psi \to \Theta)\, dE(x \leftarrow \Psi) \tag{3.3}$$

Integrating the BRDF over the hemisphere:

$$L_r(x \to \Theta) = \int_{\Omega_x} f_r(x, \Psi \to \Theta)\, dE(x \leftarrow \Psi) \tag{3.4}$$

$$= \int_{\Omega_x} f_r(x, \Psi \to \Theta)\, L(x \leftarrow \Psi) \cos(N_x, \Psi)\, d\omega_\Psi \tag{3.5}$$

Substituting the above equation 3.5 into equation 3.1, we get the result:

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{\Omega_x} L(x \leftarrow \Psi)\, f_r(x, \Psi \to \Theta) \cos(N_x, \Psi)\, d\omega_\Psi \tag{3.6}$$

- N_x: a normal surface
- Θ: an outgoing radiance direction
- Ψ: an incoming radiance direction
- $L(x \rightarrow \Theta)$: the radiances leaving point x in a direction Θ
- $L_e(x \rightarrow \Theta)$: the emitted radiances of point x
- $L(x \leftarrow \Psi)$: the incident radiances towards point x from direction Ψ
- $f_r(x, \Psi \rightarrow \Theta)$: the bidirectional reflectance distribution function(BRDF) describing the reflective properties of the surface
- $V(x \leftarrow \Psi)$: the visibility function of point x from direction Ψ.

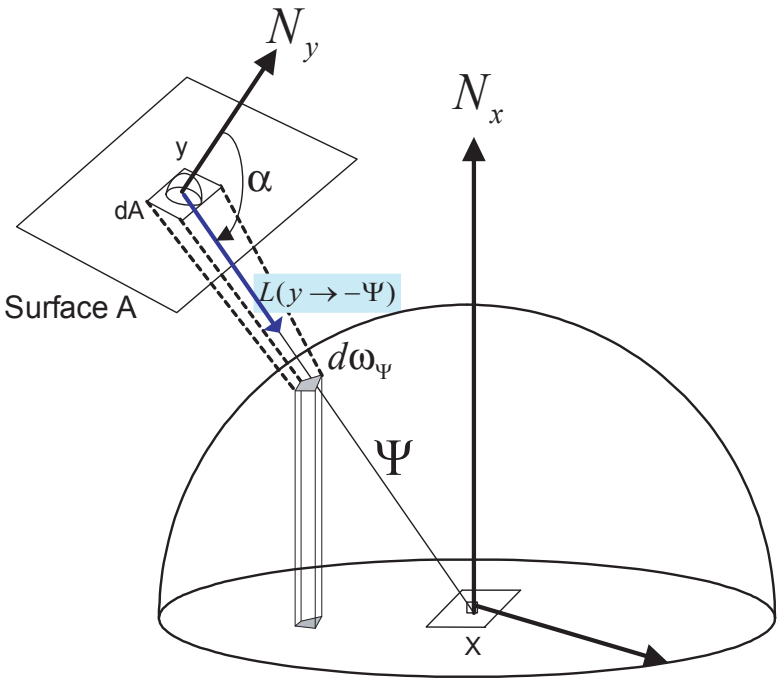


Figure 3.2: Hemisphere and Area Formulation.

Area Formulation

Instead of using the integral over the hemisphere, it is sometimes convenient to express the integral over visible surfaces from a point x. This formulates the energy balance equation in terms of area formulation as opposed to the hemisphere formulation method described above. To transform a hemisphere integral to an area integral, the differential solid angle $d\omega_\Theta$ around direction Θ is transformed to a differential surface dA_y at a surface point y. For small surfaces, an approximate solution can be used to compute the projected surface area of the solid angle subtended by a surface A (Figure 3.2).

$$d\omega_\Theta \approx \frac{A\cos\alpha}{d^2} = \cos(N_y, -\Psi)\frac{dA}{r_{xy}^2} \tag{3.7}$$

By substituting this equation into 3.1, the rendering equation can also be expressed as an area formulation, which is an integration over all surfaces:

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_A L(x \to \Psi) f_r(x, \Psi \to \Theta) V(x,y) \cos(N_x, \Psi) \left(\cos(N_y, \Psi) \frac{dA}{r_{xy}^2} \right) \quad (3.8)$$

$$= L_e(x \to \Theta) + \int_A L(y \to -\Psi) f_r(x, \Psi \to \Theta) V(x,y) G(x,y) \, dA_y \quad (3.9)$$

$$= L_e(x \to \Theta) + \int_A L_e(y_i \to \overrightarrow{y_i x}) f_r(x, \overrightarrow{xy_i} \to \Theta) V(x,y_i) G(x,y_i) \, dA_y \quad (3.10)$$

- y_i: the sampled points on the emitter
- $\overrightarrow{y_i x}$: the set of directions from the point y_i in emitter to point x
- $L_e(y_i \to \overrightarrow{y_i x})$: the radiance arriving at point x from the emitter point y_i
- $G(x, y_i)$: the geometry term between the point y_i and elements on point x
- $V(x, y_i)$: the visibility between the point y_i and elements on point x

Incoming radiance at point x from direction Ψ is the same as the outgoing radiance from point y in the direction $-\Psi$, which means $L(x \leftarrow \Psi) = L(y \to -\Psi)$ (see Figure 3.2).
The term $G(x,y)$ is the geometry relation, which depends on the relative geometry of the surfaces between points x and y.

$$G(x,y) = \frac{\cos(N_x, \Psi) \cos(N_y, \Psi)}{r_{xy}^2} \quad (3.11)$$

3.2.2 Stochastic Numerical Model for the Rendering Equation

In this section, a stochastic approach to solving a complex integral function such as the rendering equation is introduced. Then it will be shown that the two formulations can be approximated by using a numerical method.

The integral part of the rendering equation is very difficult to compute in an analytic form, which can be overcome by the use of Monte Carlo integration, a powerful technique that can be used to estimate arbitrary functions. Let $f(x)$ be a function defined over the $x \in [0,1]$, then an integral function can be defined such that:

$$I = \int_0^1 f(x) dx \quad (3.12)$$

The Monte Carlo method can be used to evaluate the integral in terms of a numerical solution by choosing N samples to estimate the value of that integral. The samples x_i are selected with the *Probability Distribution Function (PDF)* $p(x)$. The estimator $\langle I \rangle$ is:

$$\langle I \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \quad (3.13)$$

The variance of this estimate is:

$$\sigma^2 = \frac{1}{N} \int \left(\frac{f(x)}{p(x)} \right)^2 p(x) \, dx \quad (3.14)$$

As the number of samples increases, the variance decreases linearly with N. The standard deviation σ decreases with $\sqrt{N}$. Although it has a slow convergence characteristic, a number of variance reduction techniques can be adapted.

By applying the Monte Carlo technique to the two formulations in Section 3.2.1, an approximated function can be evaluated by generating N random directions Ψ_i, on a hemisphere Ω_x, distributed according to some probability density function $p(\Psi_i)$:

Hemisphere Formulation

Equation 3.6 can be rewritten in terms of a numerical method:

$$L(x \to \Theta) = L_e(x \to \Theta) + L_r(x \to \Theta) \tag{3.15}$$

$$= L_e(x \to \Theta) + \int_{\Omega} \underbrace{L(x \leftarrow \Psi)}_{\text{Incident radiance}} \underbrace{f_r(x, \Psi \to \Theta)}_{\text{BRDF}} \underbrace{V(x \leftarrow \Psi)}_{\text{Visibility}} \underbrace{\cos(N_x, \Psi)}_{\text{Geometric Term}} \underbrace{d\omega\Psi}_{\text{Solid angle}} \tag{3.16}$$

Let's denote a Monte Carlo estimator $\langle L_r(x \to \Theta) \rangle^H$ to $L_r(x \to \Theta)$ (where H stands for Hemisphere formulation), then the numerical solution to the integral is:

$$\langle L_r(x \to \Theta) \rangle^H = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{L(x \leftarrow \Psi_i) f_r(x, \Psi_i \to \Theta) \cos(N_x, \Psi_i)}{p(\Psi_i)} \right] \tag{3.17}$$

Area Formulation

In the same way, equation 3.10 can be rewritten in numerical terms:

$$L(x \to \Theta) = L_e(x \to \Theta) + L_r(x \to \Theta) \tag{3.18}$$

$$= L_e(x \to \Theta) + \int_{A} L_e(y_i \to \overrightarrow{y_i x}) f_r(x, \overrightarrow{xy_i} \to \Theta) V(x, y_i) G(x, y_i) dA_y \tag{3.19}$$

Let's denote an estimator $\langle L_r(x \to \Theta) \rangle^A$ to $L_r(x \to \Theta)$ (where A stands for Area formulation), then the numerical solution to the integral is:

$$\langle L_r(x \to \Theta) \rangle^A = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{L_e(y_i \to \overrightarrow{y_i x}) f_r(x, \overrightarrow{xy_i} \to \Theta) V(x, y_i) G(x, y_i)}{p(y_i)} \right] \tag{3.20}$$

The radiance equation can be computed in two ways as explained in equations 3.17 and 3.20. In the following section, both formulations are used in different illumination models in order to evaluate the integral function recursively.

3.2.3 Neumann Series Expansion

As explained in Section 2.2.3, the Surface Radiance Equation is written compactly using an operator form. This allows solving of the radiance equation iteratively with the Neumann series.

$$L^0 = L_e \tag{3.21}$$

$$L^1 = T \cdot L^0 = T \cdot L_e \tag{3.22}$$

$$L^2 = T \cdot L^1 = T\,(T \cdot L_e) \tag{3.23}$$

$$\ldots = \ldots \tag{3.24}$$

$$L^n = T \cdot L^n \tag{3.25}$$

$$\sum_{n=0}^{\infty} L^n = \sum_{n=0}^{\infty} T^n \cdot L_e \tag{3.26}$$

The radiance equation can be written in three terms using the above notation. The first term L^0 describes the light sources and the second term L^1 is the direct lighting. The rest of the terms, $L^2 \cdots L^n$ are indirect illumination, such that:

$$L(x \to \Theta) = \underbrace{L^0}_{\textbf{Emitter}} + \underbrace{L^1}_{\textbf{Direct illumination}} + \underbrace{L^2 \cdots L^n}_{\textbf{Indirect Illumination}} \tag{3.27}$$

$$= L_e(x \to \Theta) \; + \; \langle\, L_r(x \to \Theta)\,\rangle^A \; + \; \langle\, L_r(x \to \Theta)\,\rangle^H \cdots \tag{3.28}$$

To compute each of these terms, the Monte Carlo Hemisphere formulation and Area formulations are extended to a surface x.

For direct lighting, the area formulation equation is used (Equation 3.20)

$$L^1 \;=\; \langle\, L_r(x \to \Theta)\,\rangle^A \tag{3.29}$$

$$\;=\; \frac{1}{N}\sum_{i=1}^{N}\left[\frac{L(y_i \to \overrightarrow{y_i x})\, f_r(x, \overrightarrow{xy_i} \to \Theta)\, V(x,y_i)\, G(x,y_i)}{p(y_i)}\right] \tag{3.30}$$

The rest of the terms in Equation 3.27 are for the indirect illumination calculation. To evaluate them, hemisphere formulation is used (Equation 3.17).

$$L^2 \cdots L^n \;=\; \langle\, L_r(x \to \Theta)\,\rangle^H \tag{3.31}$$

$$\;=\; \frac{1}{N}\sum_{i=1}^{N}\left[\frac{L(x \leftarrow \Psi_i)\, f_r(x, \Psi_i \to \Theta)\, \cos(N_x, \Psi_i)}{p(\Psi_i)}\right] \tag{3.32}$$

The direct lighting calculation is extensively explained in Chapter 5, Section 5.2, and indirect illumination is described in Section 5.3.

3.3 Parametrization of Spherical Data (Solid Angles)

In this section, the various parameterization methods for representing (Hemi)spherical data will be reviewed. Our visibility data structure is heavily dependent on directional representation, as the SLHB is a collection of directional LHBs over the sphere, with associated depth values. Therefore, it is quite important to select a good parameterization for spherical data. There are many applications that require a parameterization of spherical data, and in computer graphics, a function of the (hemi)sphere associated

with values can be used in many different ways. One of the typical usages is a distribution of radiance values $L(x \rightarrow \Theta)$ on a hemisphere, which represent the radiance leaving a surface. This kind of parameterization is an environment map, or reflection map, of radiance at a surface point. In terms of the Light field technique, this holds radiance field information. Another area of implementation is as a representation of direction, as many applications use a uniform distribution of spherical directions. The BRDF is another interesting area, and there have been many attempts to formulate hemispherical data as an analytic function. Spherical harmonics and hemispherical harmonics belong to this category of research.

3.3.1 Uniform Subdivision Methods

Two uniform parameterization methods are used in this study to represent hemispherical data. The first one uses build-directional samples, by recursively subdividing platonic models. We propose another uniform parameterization method, which generates a fairly uniform distribution on the (hemi)sphere.

Uniform Subdivision of Platonic Models

The most popular subdivision of a unit sphere is to use platonic solid models as the basic function to offer a uniform subdivision. Platonic solids such as a tetrahedron, octahedron and icosahedron are used in our study to represent the directions of LHBs. Each side of the equilateral triangle is subdivided and projected to the surface of the sphere. This procedure is carried out recursively until a given depth is reached. Then the centroid or vertex points of the subdivided triangles can uniformly represent a distribution of samples on the sphere, as illustrated in Figure 3.3.

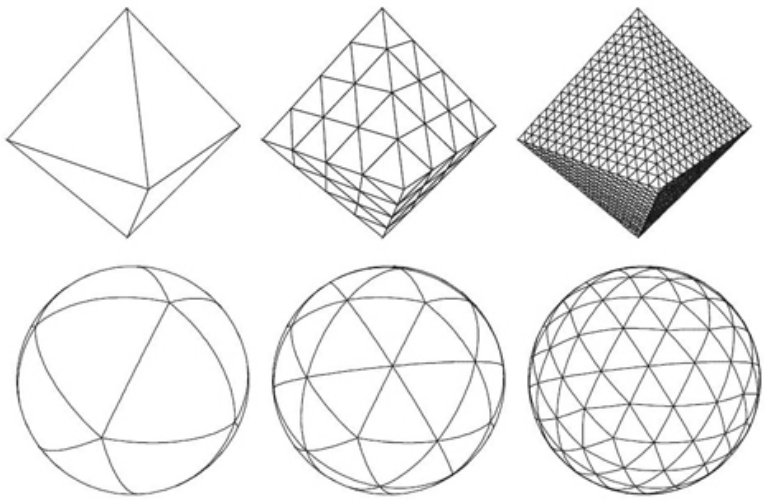


Figure 3.3: Recursive subdivision of octahedra base and icosahedron.

For instance, a tetrahedron subdivision method uniformly subdivides a quadrant into smaller triangles. The tetrahedron is recursively partitioned into smaller triangles up to a certain level to generate spherical triangles. Each point p on the polyhedron is projected onto the sphere to produce a point $p^{'}$. Applying this process to a tetrahedron L times, a solid model is generated with $4^L \times 4$. The centre (or vertex) point of each tessellated triangle represents a direction with a radiance value seen from the centre of the unit sphere. The spherical triangle can be considered as a solid angle or direction in our terms.

Tetrahedron_Subdivision ()
Triangulation
For every edge
Find midpoint of the edge
Project midpoint onto the unit sphere
For each face
Replace the face with new four triangles
Midpoints to original vertices

Table 3.1: Uniform Tetrahedron subdivision.

Uniform Spherical UV Grid

Although a simple uniform subdivision of platonic models provides fairly uniform distribution for (hemi)spherical data, there are some drawbacks in practice. One difficulty of using this method is that the number of triangles for each level increases four-fold; for example, 256, 1024, 4096, 16384, 65536 and so on. Secondly, when a hemisphere is subdivided recursively by using the uniform subdivision technique on tetrahedra, then the projection of points on the sphere onto the tetrahedron triangles generates projection errors, which results in non-uniform subdivision as shown in Figure 3.6 (a). The variance of the spherical triangles becomes larger near the center region. This error should be minimized in order to avoid uneven solid angle coverages for discrete directions. We present a new UV-Grid parameterization method to overcome these problems.

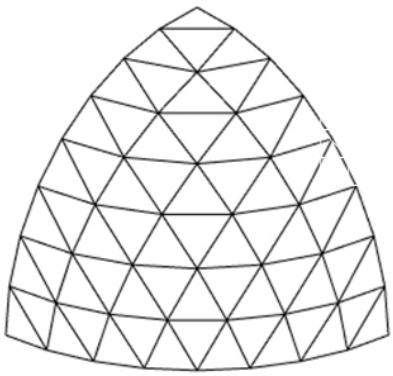
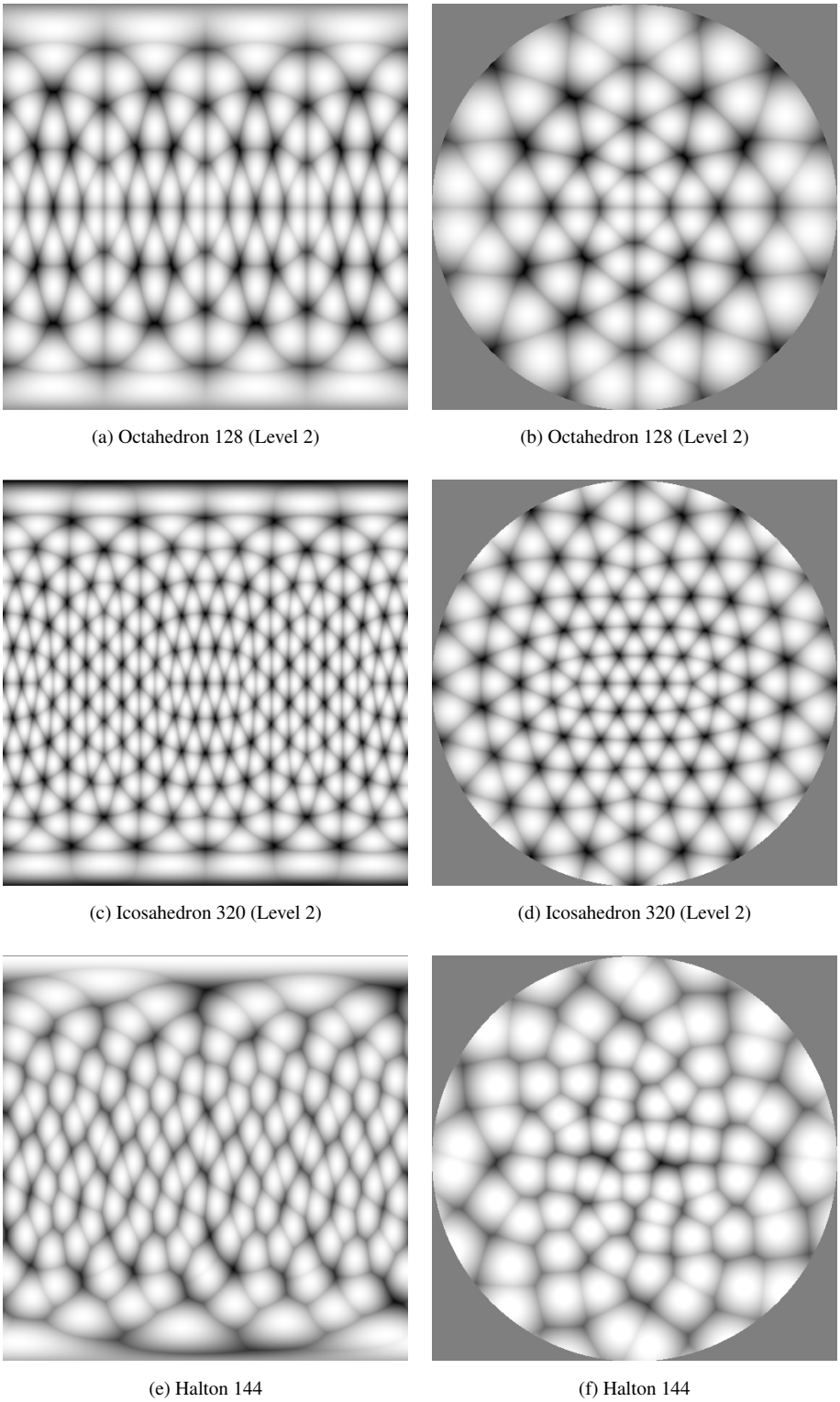


Figure 3.4: An Example of Uniform Spherical UV Grid (Quadrant).

A uniform Spherical UV grid method can be generated by simply dividing a unit sphere in a spherical coordinates system. The proposed method gives smaller variance in size, yielding uniformness over the hemisphere and more flexible subdivision level. We will denote latitude as θ and longitude as ϕ. For any given value of N, equal distance angles can be produced in latitude and longitude by dividing by N samples. The (θ, ϕ) form a direction on the hemisphere. This way, instant access to any point on

the hemisphere is possible. Figure 3.4 shows an example of this, where $N = 8$. This method produces $4L^2$ spherical triangles, where L is the subdivision level. Overall, this is a far more flexible subdivision scheme.



(a) Octahedron 128 (Level 2)

(b) Octahedron 128 (Level 2)

(c) Icosahedron 320 (Level 2)

(d) Icosahedron 320 (Level 2)

(e) Halton 144

(f) Halton 144

Figure 3.5: Solid Angle Maps for Octahedron, Icosahedron and Halton Samples.

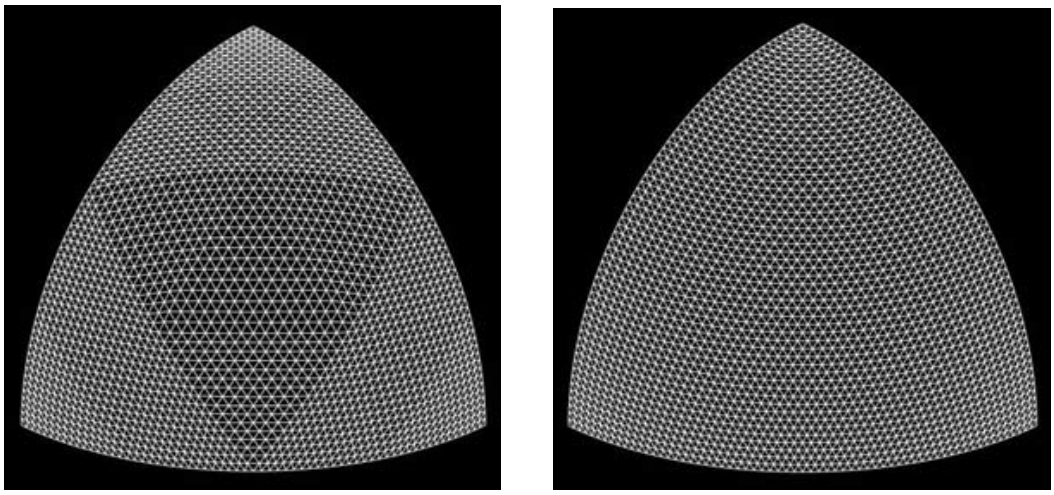
A constant query of hemispherical data was suggested by Slater [Sla02], where a portion of the

tetrahedron is rendered into an off-screen buffer, in order to query a data point distributed over the hemisphere in constant time. A weakness of this technique is the limited size of the off-screen buffer memory to handle large data points in a commodity graphics card. However, our method does not require more than a few hundreds of directions, so that we can adapt the off-screen technique to query the directional data in constant time.

Apart from the uniform subdivision methods, we also employ a low discrepancy sampling scheme, like a Halton sequence. In order to generate uniform directions, random samples in 2D cartesian coordinates are transformed to spherical coordinates to form samples in a sphere. In this way, any number of uniformly directions on a sphere can be generated as shown in Figure 3.5 (c).

3.3.2 Evaluation of Uniform Subdivision Methods

Figure 3.6 illustrates the difference between the tetrahedron subdivision method and spherical UV Grid method. The tetrahedron method was used in the previous [SMKY04] study to query a random position on the hemisphere. IT can be seen from the figure that the tetrahedron subdivision produces uneven tessellated triangle areas, and the variance is very high compared with our proposed solution. The spherical UV Grid shows that the area of each element is more evenly tessellated.



(a) Tetrahedron Subdivision (b) UV-Grid Subdivision

Figure 3.6: Comparison of Two Uniform Subdivision methods.

The proposed Spherical Layered Hit Buffers are a discrete representation of the continuous visibility field. Therefore, a limited number of discrete directions can result in banding artifacts. This is discussed and a Cone Approximation technique is proposed in Section 5.3.1. The Cone Approximation can approximate Monte Carlo integration with only a small number of LHB directions.

Figure 3.7 shows a profile of the tetrahedron subdivision method, where 4 Levels of subdivision have been chosen to show the effect of angle variance. The number of directions for each graph is 8 (Level 0), 32 (Level 1), 128 (Level2), an 512 (Level 3). Each graph represents a profile of angular variance around the equator. The x-axis is in degrees between 0 and 360. The Y-axis is cosine θ, between the accurate discrete direction and the estimated directions. In other words, if the visibility enquiry

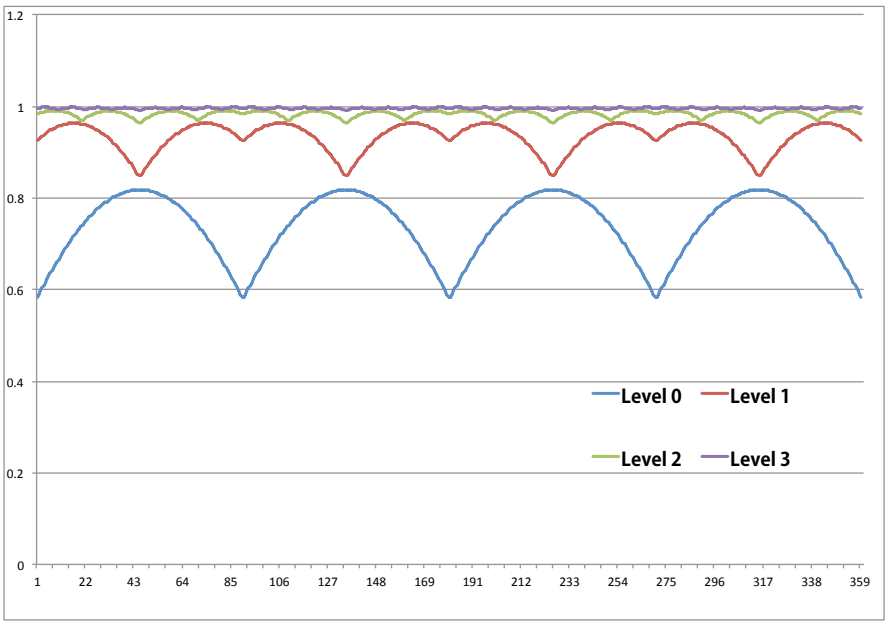


Figure 3.7: A Profile of Tetrahedra in terms of Solid Angle. The X-axis is 0°-360° around the equator, Y-axis is a cosine angular difference.

direction is perfectly aligned to the discrete direction, then the cosine angle is 1, or 0 degree difference. Therefore, any direction between the discrete delta directions has a cosine fall off characteristic. Level 3 subdivision of tetrahedra shows a very low angle difference, whereas Level 0 shows that the cosine angle is below 0.6 (or 53 degrees) in the worst case. This means that the angle difference between the accurate direction and the worst case direction could be 53 degrees. Therefore, it is important to choose a high number of directions, which ensures the angle variance is a minimum while achieving real-time rendering. Again, we employ a cone approximation to resolve this problem in Section 5.3.1. Figure 3.5 illustrates the solid angle map of various parameterizations, including Octahedron, Icosahedron and Halton sampling.

3.4 Spherical Layered Hit Buffers

The key idea behind our real-time rendering algorithm is to solve the rendering equation using an approximated visibility function, especially for indirect lighting computation. For this reason, we propose *Spherical Layered Hit Buffers* (SLHB), which represents a 5D visibility function (position (x,y,z) and direction (θ,ϕ)) in multi-layered buffers, called *Layered Hit Buffers* (LHB). This structure can be thought of as a multi-layered orthographic projection in many directions. A 2D illustration of directions is shown in Figure 3.8. The LHB directions resemble the PSF directions in our early work [SMKY04]; however, the SLHB does not store nor propagate any radiances.

Given a scene in real-world coordinates, a translation, scale and rotation is applied to fit the scene into a unit cuboid, centered at the origin as shown in Figure 3.9. Then, the scene can be orthographically projected onto the frame buffer using a hardware rasterization. The whole process of rasterizing the scene into LHBs can be defined as a single matrix form, as given in Equation 4.1. Unlike the conventional

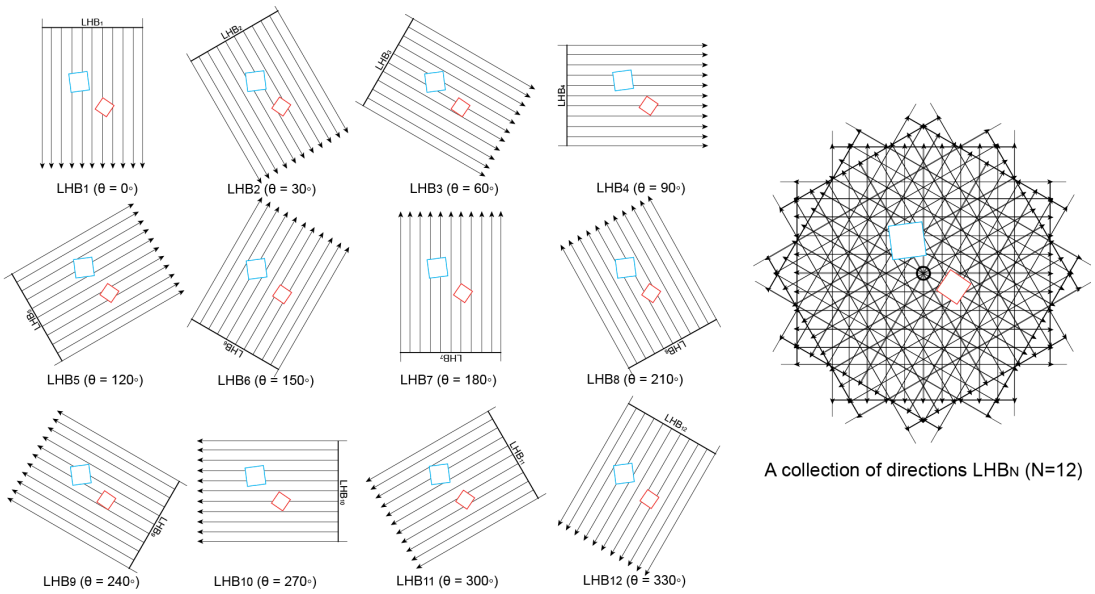
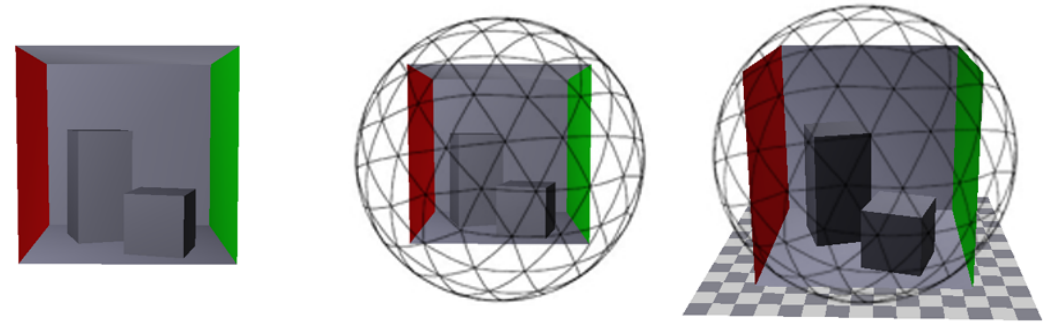


Figure 3.8: A 2D example of multiple directions to build a SLHB.



(a) Scene in world coordinates (b) Scene with LHB directions (Icosahedron) (c) A canonical LHB for rasterization

Figure 3.9: Layered Hit Buffers in icosahedron directions.

rasterization method, a CUDA Deep rasterization method is proposed to construct a LHB in a canonical view. The CUDA Deep rasterization outputs a list of all intersected polygon IDs and depth values in the Z-axis direction. In other words, each fragment location at (i, j) in the LHB stores a list of elements that are defined with a depth value and associated polygon ID. In this way, a partial visibility field is constructed in a single orthographic view. The LHB structure shares some similarity to the A-buffer [Car84], which contains a list of elements. However, fixed pipeline graphics hardware is not capable of writing a structural output in random global memory locations. In order to simulate this process in the graphics hardware, some depth-peeling algorithms can be adapted; however, multiple rendering passes prohibit real-time rendering. In contrast, our proposed CUDA Deep rasterization method is a fully customized rendering pipeline, which can output any structural data and also fully benefits from GPU parallel processing. The details of building the LHB using a CUDA rasterization method is given

in Chapter 4. The LHB structure has some similarities to Layered Depth Images [SGHS98], where the LDI stores radiance values and the LHB stores primarily depth information to capture the visibility field. A single rasterization can build a LHB with a size of $N \times N$ fragments in an orthogonal view. The SLHB is a collection of LHBs in many directions (θ, ϕ), in order to capture the visibility in different views. When the scene in a unit cuboid is rotated within (θ, ϕ), a new canonical visibility field can be rasterized to build $LHB(\theta, \phi)$. We chose l discrete directions to represent the direction of the LHB. Finding the closest direction to any arbitrary direction can be achieved in a constant time using a pre-computed *Directional Map* in a similar way to [Sla02], where each pixel in the textures stores a nearest direction. The directions of the SLHB should follow a uniform distribution over a sphere, such that the angular variance between two nearest directions is kept to a minimum. In the following section, a few parameterization methods will be examined, which offer uniformly distributed directions.

3.5 Summary

In this chapter, the mathematical foundations of global illumination were introduced. The rendering equation can be analytically solved using a Monte Carlo integration technique, and more details on the implementation of real-time rendering methods are explained in Chapter 5. We also proposed an acceleration structure, based on SLHBs, which is a 5D visibility field for instant occlusion query. More details on how to generate the LHB using a CUDA deep rasterizer will be explained in Chapter 4. Several parameterization methods are presented to represent the directions of the SLHB. An analysis of the distribution of spherical data in terms of solid angles was discussed in this chapter.

Chapter 4

CUDA Deep Rasterization

4.1 Introduction

In the past decade, significant advances in graphics hardware technology has provided a flexible environment for dealing with complex tasks. In computer graphics, the graphics or rendering pipeline refers to rasterization supported by commodity graphics hardware. The graphics pipeline takes the vertex data of a 3D scene as the input, which rasterizes objects into a screen size frame buffer as the output. There are two industry standards, which are *OpenGL* and *Direct3D*, and they offer similar graphics pipeline models. Traditionally, the graphics pipeline was a fixed-function on hardware, however recent developments have provided increasing programmability with *vertex shaders*, *geometry shaders* and *pixel (fragment) shaders*.

A shader is a program to calculate some rendering effects on the graphics hardware. A fixed-function pipeline allows only geometry transformation and pixel-shading, whereas shaders offers high degree of flexibility on Graphics Processing Unit (GPU). In the early development stages, only pixel shaders were programmable; however as the GPU evolved, vertex and geometry shaders were introduced. A vertex shader is applied to each vertex to transform the data in object space to that in view space. The output of the vertex shader feeds to the geometry shader, which is responsible for tessellating the triangles into refined vertex data. The output positions are then rasterized by interpolating within three vertices to form pixels within its area. A pixel shader is applied to each pixel to compute the screen color. Modern graphics units have hundreds of stream processors running concurrently to compute shading operations in a massively parallel way.

Despite significant advances in graphics hardware, there are still some limitations. In particular, the output of the pixel shaders is bound to a fixed sized texture rather than a user defined data structure. In our new algorithm, it is the aim to use graphics hardware to build a structural buffer, which contains a list of primitives with depth information per pixel, in a single pipeline. For this reason, a traditional graphics pipeline is not able to deliver scatter writing to build structural data output. Therefore a fully customizable rendering system is proposed, which uses the GPU as a general parallel processing unit. Shading languages such as *OpenGL* and *Direct3D* are implemented for 3D graphics requirements, whereas the CUDA architecture is employed for general purpose parallel processing. The standard pipeline is replicated in a CUDA architecture, and the entire pipeline is run in a programmable graphics hardware. The

proposed graphics pipeline has been designed and modified to fit the intended need and also run cache efficient manner as illustrated in Figure 4.1. CUDA allows to manage constant and shared memory for caching purpose, which is as fast as the registers. The performance of the *CUDA Deep rasterization* method is comparable to standard OpenGL for moderately complex models, and for complex models with many micro-polygons, this new rasterizer achieves twice the speed of OpenGL functions. In the following section, the details of the CUDA architecture is introduced.

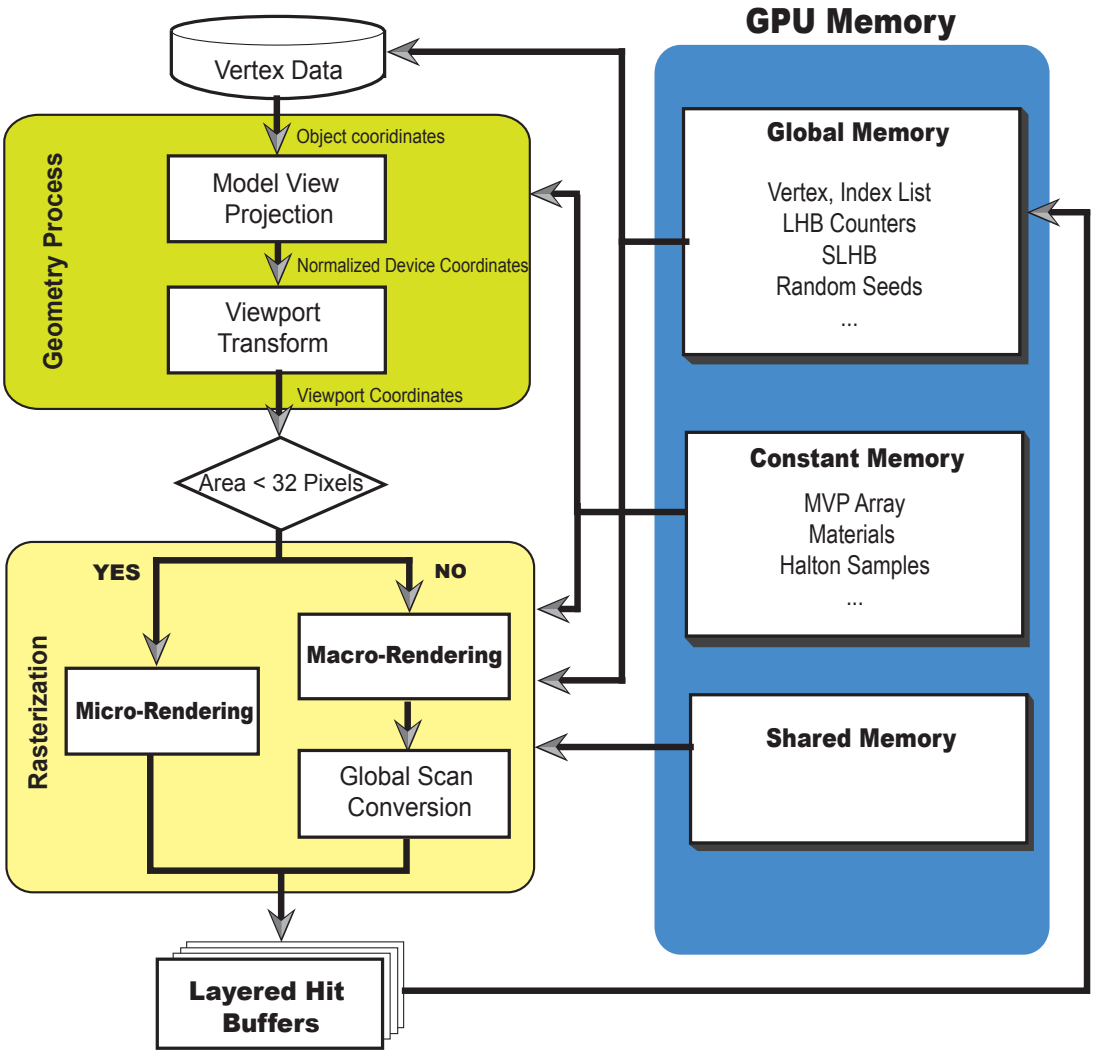


Figure 4.1: Overview of CUDA Deep Rasterization to build Layered Hit Buffers (LHB).

4.2 CUDA Architecture

In the past few years, the programmable graphics processor has evolved dramatically into highly parallel multi-core processors, which can support many multi-threads with high memory bandwidth. This results in massive computational power compared with CPU processors as illustrated in Figure 4.2. The reason behind this evolution is that the GPU can be specialized and optimized for highly parallel computation, by simplifying the number of instruction sets and control flows.

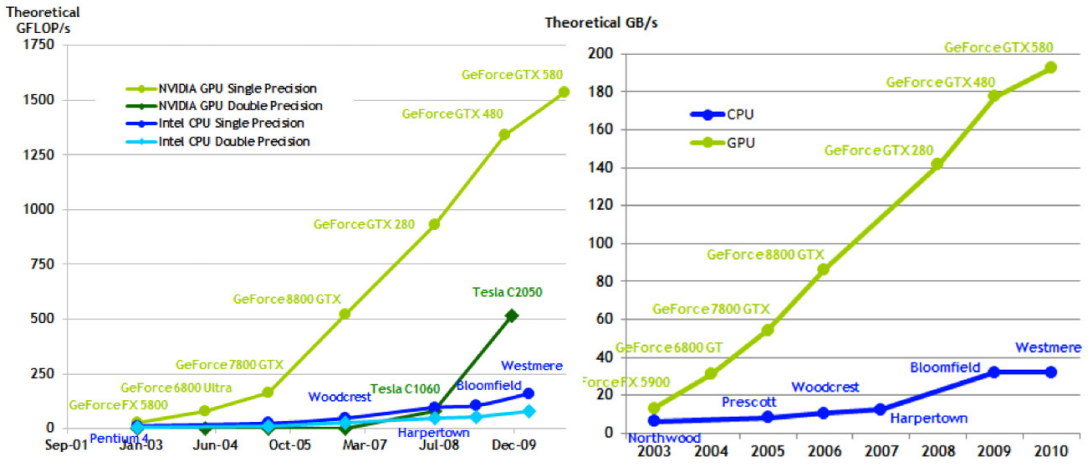


Figure 4.2: Performance Comparison and Memory Bandwidth for GPU and CPU (from NVIDIA [Cud11]).

In 2006, NVIDIA introduced Computed Unified Device Architecture (CUDA) as a general purpose computing architecture, which offers hundreds of arithmetic units in a GPU as illustrated in Figure 4.3. The CUDA is a new parallel programming model and provides instruction sets offering a high-level programming language to control the parallel architecture easily. It allows users to access the GPU directly in data parallel ways, and this has been chosen over DirectCompute or OpenCL language for this project, because it allows low level processing of graphics hardware with high-level programming languages like C/C++. CUDA also provides functions that give better controls for the graphics hardware kernels.

Figure 4.3 illustrates the architecture of the NIVIDA GTX 480, which is the graphics card that is used to implement the methods proposed in this thesis and measure the results timing. The NIVIDA GTX 480 features 15 *Streaming Multiprocessors* (SM). Each SM (on the right-hand image) utilizes 64KB of *Shared Memory* / L1 cache on the chip between 32 CUDA cores. In other words, NVIDIA GPUs are based on multi-processors with a *Shared Memory* architecture. There are 480 CUDA cores running in a Single Program Multiple Data (SPMD) model. The CUDA cores execute the same instructions simultaneously, as each SM is a highly parallel multiprocessor, supporting up to 48 warps simultaneously, where each warp is 32 threads. Therefore, GPUs can maintain up to 1536 threads per stream multi-processor. The NIVIDA GTX 480 Fermi architecture features four memory types, which are *Global (Device) Memory*, *Local Memory*, *Shared Memory* and *Constant Memory*. The *Global (Device) Memory* is accessible by all Streaming Multiprocessors, but has very high latency (400- 800 cycles). The *Local memory* is also slow and uncached, but only accessible within each SM. The *Shared Memory* is expected to have very low latency and high throughput, which is located in the L1 cache near each processor core. The shared memory is also known as a user defined cache, because it is configurable to support caching of local and global memory. The *Constant Memory* allows read-only access and provides faster access than the global memory.

Figure 4.3: NVIDIA GTX 480 Architecture(left) and a Stream Processor(right) (from NVIDIA [Cud11]).

4.2.1 CUDA Implementation Issues

OpenGL or DirectX APIs are able to read scatter data from texture memory, but they are unable to write arbitrary addresses in the memory. In contrast, CUDA is capable of accessing both scattered read and write. Because of this benefit, CUDA has been chosen over the traditional fixed graphics pipeline in order to produce a structural output directly into GPU memory, within a single rendering pass. In order to build an efficient rasterization in CUDA architecture, there are a few things to consider.

The GTX 480 graphics card features 15 Stream Multiprocessors with 48 warps, so it can run 23,040 threads (15 SM x 48 warps x 32 threads) at the same time. In order to maximize the scheduling of stream processors, at least 2-3 times more than the 23,040 thread jobs must be allocated to CUDA cores, in order to hide the latency. Therefore, it is very important to avoid any thread serialization and allocate well designed parallel jobs to the multi-processors. This issue is addressed in Section 4.4 to show how to allocate thread jobs for micro and macro polygons.

Regarding memory management, any data transfer between the CPU and GPU should be minimized, because the transfer rate is extremely slow. The local and global memory is in the device memory, whereas the shared memory and the constant memory are in the on-chip cache memory. Access to the shared memory is extremely fast and highly parallel, and is generally hundreds of times faster than the local or global memory. Therefore the new algorithm efficiently utilizes the shared memory as user managed caches, to hold the portion of global information.

Figure 4.1 shows the GPU memory structure of our algorithm. We have allocated scene data, the SLHB data structure and random variables in the global memory. For faster access, read-only data such as Model View Projection Matrices, Material properties and Halton sequences are stored in the constant

memory. The shared memory is heavily used in the rasterization method to store intermediate data and and load global data.

4.3 Building Layered Hit Buffering using CUDA Rasterization

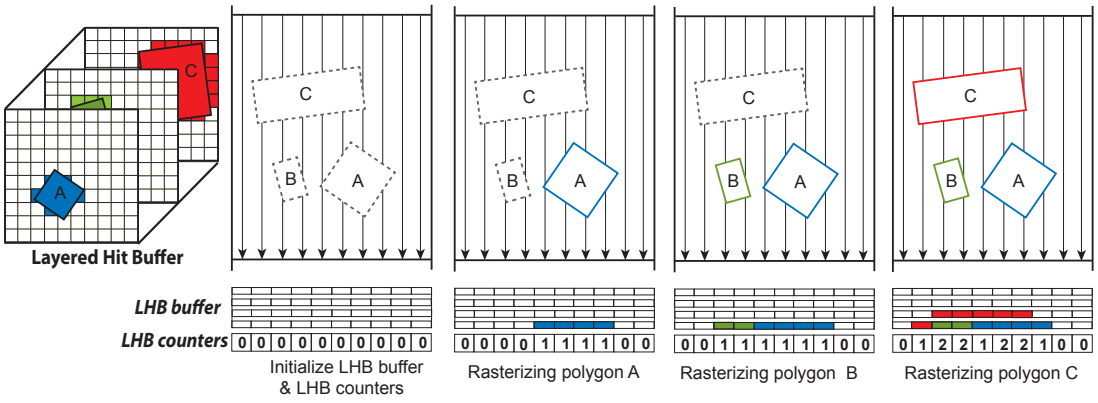


Figure 4.4: An example of CUDA Deep Rasterization to build a Layered Hit Buffer (LHB). Polygon A,B and C are rasterized to the LHB buffer.

The aim of the proposed rasterization is to construct a list of polygon IDs and depth values along pixels, at the intersection positions where rays hit objects for a given direction. Using graphics hardware, this process can be carried out efficiently by employing a rasterization method rather than a ray-casting algorithm.

Figure 4.4 shows a rasterization process of polygons into Layered Hit Buffers. An LHB is constructed by rasterizing the scene into memory buffers in an orthogonal projection. Each pixel in the orthogonal view represents ray-polygon intersections. Therefore, we store a list of elements that consist of the depth and id information along the ray at all intersection positions. Ideally, the list of elements is stored in a linked list data structure. However, for simplicity, we have opted to use a fixed memory buffer with a counter per pixel. Thus, when the number of intersections at a particular location is higher than the fixed element size, the element information is discarded. The counters are used to define the location of elements at each intersection. They are initialized with zero value at the beginning.

When the first polygon *A* is rasterized, the counter is incremented and the element information (Depth and Polygon ID) of front facing polygons is stored in the LHB at the zero location. Due to thread parallel processing, there is a concurrency control problem involved in synchronizing the LHB. In many multi-threaded programming, a race condition arises when multiple threads attempt to access the shared data and write to the same location at the same time. Kurt et al [DDSC11] suggested a *Wait-Free* mechanism to synchronize accesses to shared memory data. However, CUDA provides *Atomic Operations*, which deal with read- modify-write (RMW) operations. For instance, we employ the *atomicInc* function to read the 32-bit word counter and increment its value, then store the result back to the same address. These three operations are performed in a single atomic instruction without interference from other threads. Therefore, *atomicInc* increases the counter by one in order to reserve memory for current

element location data, then stores the element information in the previous entry of the array. Next, the polygons *B* and *C* are rasterized in same way and the counters are incremented as the elements are appended to the list. Each element has two values; one is of 32 bits floating point depth value, and the other one is the polygon ID in a 32-bit word. In order to save storage space and increase the bandwidth, it is possible to pack the polygon ID and depth values into a 32 bit integer for a simple scene. For instance, polygon ID is stored in 20 of the bits and the integer depth value in 12 bits. In this way, the memory requirements for the LHB can be minimized and the bandwidth made more efficient by reducing the output to global memory, due to its very high latency.

The CUDA Deep Rasterization constructs an LHB buffer for a given direction. This process is applied to *N* spherical directions to build the Spherical Layered Hit Buffers (SLHB), which serve as a 5D visibility field for the proposed real-time rendering method.

4.4 CUDA Deep Rasterization

In this section, a Deep Rasterization method is presented, which is designed for the fast execution of many parallel instructions simultaneously. First, there is an overview of the related works on rasterization methods and their limitations. Then a new CUDA-based rendering pipeline is described, following by more detail of the model view transform. Micro and macro-rasterization methods to output projected polygons into LHBs is then presented.

4.4.1 Overview

Fatahailan [FLB+09] proposed an algorithm that rasterizes micro polygons for efficient defocus and motion blur using CUDA. Several researchers employ REYES (Renders Everything You Ever Saw) rendering methods [PO08], [ZHR+09] on GPUs for realistic rendering. Although these methods can produce high quality results, due to their sub-pixel level accuracy, they are not suitable for real-time applications. The new Deep Rasterization method proposed in this dissertation is essentially aimed towards building a Layered Hit Buffer (LHB) that contains a list of depths and polygon IDs at all intersection points per pixel, from an orthographic projection view. For this reason, the Deep Rasterization method closely resembles the A-Buffer algorithm [Car84]. Spherical Layered Hit Buffers are a collection of LHBs in carefully chosen spherical directions. In order to implement this on conventional fixed pipelines, the hardware requires many rendering passes to capture the fragment data per pixel, using a depth-peeling algorithm [Mam89]. The K-Buffer [BCL+07] uses multiple rendering targets to capture many layers in a single pass. Myers et al [MB07] introduce a technique to employ MSAA (multi-sampling anti-aliasing) with the stencil test, which allows 8 fragments per each geometry rendering pass. Bucket depth peeling [LHLW09b] captures 32 fragments for a single pass. All these methods are limited by the maximum render target, so that a complex area in a scene still requires many rendering passes to build an array of fragment data. In contrast, the new Deep Rasterization method has no limit to writing structure data in Layered Hit Buffers. Another benefit of using CUDA rasterization is that there is no need to assign buffers to textures, since LHBs are directly stored in the GPU global memory.

A list of elements in a LHB should ideally be sorted in depth order, such that the nearest intersected

element can be retrieved efficiently. However, a sorting algorithm in post-processing takes more than a simple linear search of the elements when an average number of elements is small. In order to build Spherical Layered Hit Buffers, a CUDA deep rasterization is applied for many spherical directions in orthographic views. In this way, the SLHB forms a 5D visibility structure, thereby maintaining depth and polygon information for each element. Our CUDA rasterization is inspired by Liu et al [LHLW10] work, who showed that CUDA could be used as an alternative platform to the fixed function method. In contrast, our solution is highly optimized, achieving this at least 4 times faster than their solution. Also their method does not work well on a scene where a few polygons occupy large portion of area in screen space resulting in unreasonably low frame rates for a very simple scene such as a cornell box. In contrast, we deliver an efficient parallel processing method by adapting two rendering methods; one for micro polygons and the other for macro polygons. In this way, the proposed CUDA rasterization method is able to deliver high frame rates for rasterizing in a mixture of micro and macro polygons.

4.4.2 Rendering Pipeline and Memory Structure

The fundamental procedure of a GPU pipeline is to receive a group of polygons and perform all necessary operations, and then output pixels. The first step in optimizing the rendering pipeline is to find a way to store the geometry data in the GPU memory efficiently. It is common in data parallel programming to use Struct of Arrays (SOA) data over Array of Structs (AOS), which also applies to CUDA architecture. Thus polygon vertex data in SOA is initially stored in the CPU and transferred to the GPU once during the whole rendering period. Although GPU device memory is faster than CPU memory access, the latency to transfer the GPU memory to the CUDA cores takes typically 200-300 cycles, so this process should be is minimized by allocating sufficient independent arithmetic instructions to threads.

The proposed CUDA Deep rasterization method does not require computation of any shading process, because it only needs to construct a visibility structure such that normal color and texture are discarded in the the pipeline. In this way, the data loading from the GPU device memory can be minimized in order to rasterize multiple scenes into LHBs efficiently. Therefore, the rendering pipeline focuses on optimizing a geometry process and a scan conversion process as shown in Figure 4.1. Since there are tens of thousands of threads available to compute the data simultaneously, it is natural to allocate a triangle for each thread. A single thread takes three vertices from the global memory and multiplies it by a Model View Projection (MVP) matrix to obtain the projected coordinates in Normalized device coordinates (NDC). An MVP matrix is pre-computed for each spherical direction (θ, ϕ) and stored in *constant memory* for instant access with no latency.

$$\mathbf{MVP}_{(\theta,\phi)} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\textbf{Orthographic Projection}} \underbrace{\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\textbf{Rotation Matrix } \phi} \underbrace{\begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\textbf{Rotation Matrix } \theta}}_{LHB(\theta,\phi)} \underbrace{\begin{pmatrix} S.x & 0 & 0 & T.x \\ 0 & S.y & 0 & T.y \\ 0 & 0 & S.z & T.z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\textbf{Uniform Translate, Scale Matrix}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \tag{4.1}$$

It is straightforward to construct the MVP matrix. Firstly, the scene in world space is scaled to a unit cuboid, then moved to the origin in order to ensure the viewport covers the whole scene. In this way, the visibility field holds multiple layered depth structures in any direction. In order to transform objects, *translate, scale and rotation* functions are applied to vertex data as in equation 4.1.

Orthogonal Projection

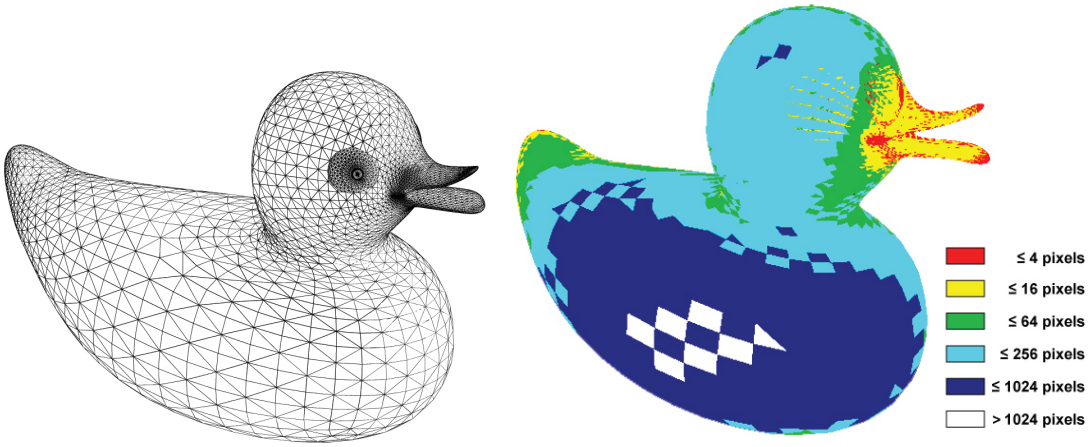
Unlike A-Buffer rendering, which uses perspective projections, an orthogonal projection is used to build orthographic linear depth buffers. Constructing a matrix for orthographic projection (Equation 4.2) is much simpler than using a perspective projection matrix. All x, y and z component in eye space are linearly mapped to NDC (Normalized Device Coordinates). The orthographic transform is given by the following matrix:

$$\mathbf{P}_{\textbf{orthogonal}} = \underbrace{\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\textbf{Orthographic Projection}} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\textbf{Orthographic Projection}} \tag{4.2}$$

The frustum is translated so that its center is at the origin, then it is scaled to the unit cube. Orthographic projection is predefined in the viewing volume of $l = -1, r = 1, t = 1, b = -1, f = -1$ and $n = 1$. Clip coordinates are divided into w perspective divisions to generate coordinates in the NDC space. The range of values is normalized between -1 and 1 in all axes. Then the viewport transform converts the coordinate in viewport space and the windows coordinates are finally passed to the rasterization process. The whole geometry process is done by a single matrix multiplication, with a few additions, which can be very efficiently processed in a parallel manner.

The rest of the process is aimed at optimizing the scan conversion of the projected triangle in a GPU friendly way. Once windows coordinates are calculated, the bounding box of the triangle is obtained to decide whether a minimum projected area is micro or macro polygon.

Figure 4.5 illustrates a typical scene with a multi-colored index to show the area of projected triangles. We opt to choose 32 pixels as a cut off to distinguish the micro and macro regions since it is a warp size in CUDA. The scan conversion process for a micro polygon in an area of under 32 pixels is efficiently computed on the fly. Otherwise a two-step process macro rasterization is applied for large



(a) Wireframe (b) Polygon Color Chart

Figure 4.5: Micro and Macro Polygons.

polygons. The following sections will describe how the rasterization process is organized in the micro and macro rasterization methods.

4.4.3 Micro-Rasterization

Half-Space Rasterization

Parallel processors, such as CUDA, benefit from the half-space rasterization method, which requires minimum setup. The proposed micro CUDA rasterizer is based on a barycentric scan conversion method. Figure 4.6 shows an example of the half-space of a triangle. One side of the line is positive and the other is negative, therefore it splits the space in half, and the location where all three edges are positive indicates the inside of the triangle. For any circumstance where any of the three half-space functions is negative, then it is outside of the polygon, and when a half-space function is zero then it is on an edge. Therefore, it can be determined whether a pixel is inside or not by evaluating the half-space functions at the pixel center.

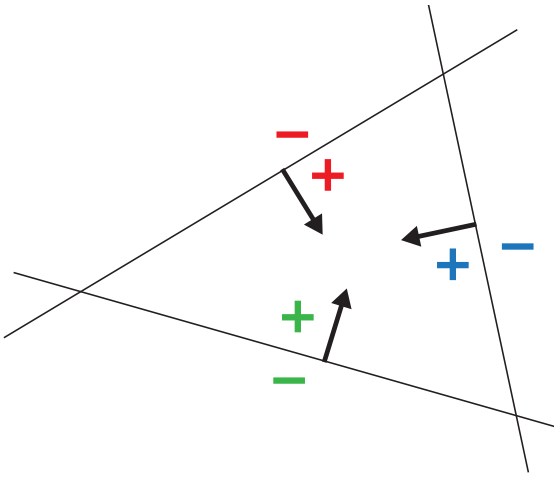


Figure 4.6: Half space rasterization method.

Listing 4.1: Simple Half Space Rasterization.

```
1  // Trinalges is defined by three vertices (x1, y1), (x2,y2) and (x3, y3)
2  // Find Bounding Box of a triangle
3  int minx, max, miny, maxy;
4  for(int y = miny; y < maxy; y++) {
5      for(int x = minx; x < maxx; x++) {
6          // When all half−space functions positive, pixel is in triangle
7          if( (x1−x2)∗(y−y1) − (y1−y2)∗(x−x1) > 0 &&
8              (x2−x3)∗(y−y2) − (y2−y3)∗(x−x2) > 0 &&
9              (x3−x1)∗(y−y3) − (y3−y1)∗(x−x3) > 0) ) {
10             int list_location = atomicInc( LHB_Counter [x][y] )
11             LHB [x][y][list_location] = {Depth, PolyID} // store an element
12         }
13     }
14 }
```

List 4.1 describes a minimal implementation of the half-space rasterization method. First, the surrounding bounding box of a triangle is found, then the half space parameters A, B, C and computed. The triangle vertices are in counter-clockwise order such that all the positive values represent the inside of triangle.

For any given point (x, y) when computing the half-space functions, if $(A > 0$ and $B > 0$ and $C > 0)$ then the point is inside of the triangle; otherwise it is outside.

$$\begin{aligned} A &= (x1 - x2) * (y - y1) - (y1 - y2) * (x - x1) \\ B &= (x2 - x3) * (y - y2) - (y2 - y3) * (x - x2) \\ C &= (x3 - x1) * (y - y3) - (y3 - y1) * (x - x3) \end{aligned} \tag{4.3}$$

Unfortunately, this basic implementation is not optimized at all. Instead, an incremental method is suggested by applying delta value to both x and y directions such that the scan conversion computation involves only a few additions and comparisons. An optimized Micro Rasterization method is given in List 4.2. In order to efficiently compute depth values for each pixel, half-space parameters are adapted to find the delta slope of the edge. Further optimization of depth values can be achieved by using delta additions, however CUDA floating point multiplication is as efficient as addition or subtraction arithmetic. In order to avoid any holes in the scan conversion function, careful consideration must be given to precision and sub-pixel accuracy.

The setup cost of the half-space micro rasterization per triangle is very low compared to the scan line conversion algorithm. The micro-rasterization method is not ideal for large triangles, since the thread can be serialized to process a single job while some other threads finish the scan conversion of small regions. Therefore, a macro-rasterization method is recommended for larger triangles.

Listing 4.2: Optimized Half Space Micro Rasterization with Depth Computation.

```
1   float A = (z3 − z1) ∗ (y2 − y1) − (z2 − z1) ∗ (y3 − y1);
2   float B = (x3 − x1) ∗ (z2 − z1) − (x2 − x1) ∗ (z3 − z1);
3   float C = (x2 − x1) ∗ (y3 − y1) − (x3 − x1) ∗ (y2 − y1);
4   // back−face culling
5   if(C <= 0) return;
6   float AC = −A/C, BC = −B/C;
7   float3 Dx = X − float3(x2, x3, x1);
8   float3 Dy = Y − float3(y2, y3, y1);
9   float3 Dxy = (Dy∗X − Dx∗Y);
10  float3 Cx, Cy = Dxy − (Dy∗MinX) + (Dx∗MinY);
11  float Z = z1 + (MinX − x1)∗AC + (MinY − y1)∗BC;
12
13  if (Area >= 32) then process Macro polygon otherwise half−space rasterization below
14  for(int y = 0; y < iHeight; y++, Cy += Dx )
15      for(int x = 0, Cx = Cy; x < iWidth; x++, Cx −= Dy)
16          if(Cx.x <= 0 && Cx.y <= 0 && Cx.z <= 0) {
17              Depth = Z + AC∗(x) + BC∗(y);
18              PolyID = Global Thread ID;
19              int list_location = atomicInc( LHB_Counter [x][y] )
20              LHB [x][y][list_location] = {Depth, PolyID}
21          }
```

4.4.4 Macro-Rasterization

In this section, three rasterization methods are presented for macro polygons, namely 4x4 Block Half-Space Rasterization, Linear Bock Rasterization and Edge-Table Rasterization methods. As mentioned previously, each thread processes a single triangle independently in a parallel manner. Therefore it is important to allocate a similar work load to each thread. However, the half-space rasterization method for a large polygon could take a significant amount of time to process a single polygon compared to micro polygons. Therefore, the fundamental basis for processing macro polygons is to divide them into small segments such that more threads can be allocated to finish the task. Unlike the Micro-Rasterization method, Macro-Rasterization employs a two-step *divide and conquer* algorithm. Once a triangle is identified as a macro polygon, the first step is to divide the triangle into small 16 pixel regions, and compute and store some parameters in the global memory. In the second step, a thread is allocated to each small segment to finish the scan conversion task with pre-computed parameters. The small segments of the polygon shares the same parameters required to fill the polygon so *shared memory* is used to boost the performance for the same task.

The underlying idea of the macro-rasterization method is to efficiently allocate thread processors to every pixel without causing a serialization problem. Three proposed methods for this are explained in

the following sections.

4x4 Block Half-Space Rasterization

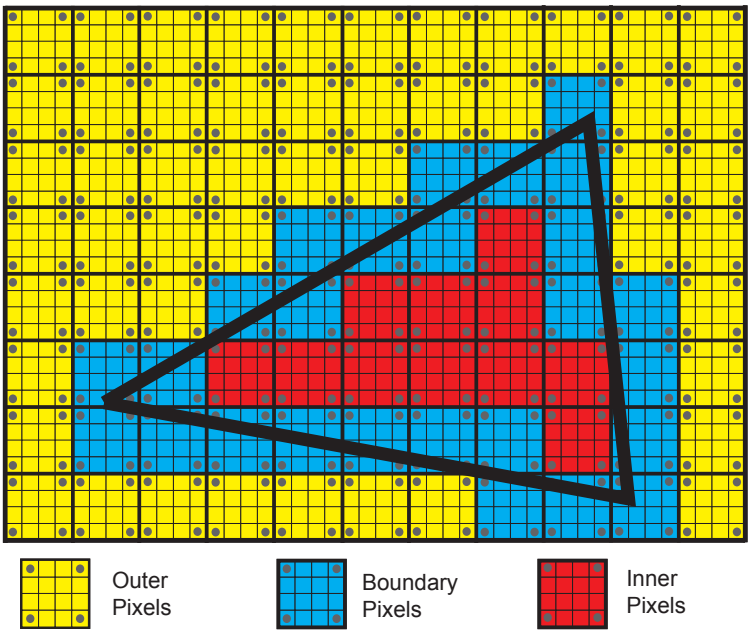


Figure 4.7: 4x4 Block Half-Space Macro-Rasterization.

The first method is an extension to the half-space rasterization technique. In the first step, instead of evaluating every pixel, blocks are identified and assigned to one of three categories; Inner blocks, Outer blocks and Boundary (Edge) blocks. This method detects whether a 4x4 block is fully covered outer block, not-covered inner block, and partially covered edge block as shown in 4.7. Inner blocks will fill the whole region without half-space evaluation, Outer blocks will be rejected, and partially covered Edge blocks will only be evaluated for every pixel in the second stage. This method evaluates on all four corners of the blocks. The fully covered inner block area will have 4 positive half-space values, not-covered outer blocks will have 4 negative values, and partially covered blocks will have at least one positive value. Therefore, only a small percentage of the blocks are fully evaluated, where the half-space functions on the outside and inside blocks are skipped. However, this method cannot handle the partially covered block where a small triangle is located at only inner part of the block without touching four corners of the block. The reason that 4x4 pixel sized blocks are chosen, is that they are in a half warp size, such that 16 evaluated pixels can fit perfectly to a half warp thread processors.

In the second step, the four corner values that were computed in the first step are used to quickly determine the z value by linear interpolation. Since memory access is more localized in a block of 32(or 16) pixels, the shared memory is used to load data efficiently and to evaluate half-space functions. Therefore, in the first pass, only block coverages are computed. If blocks are fully covered or partially covered, then block data with associated parameters are stored in a global queue to be used in the second-pass. The second pass waits until all thread processors in the first pass are finished before outputting partially or fully covered blocks. In the second pass, each block is allocated with 16 threads such that

each pixel is processed with a single thread, in order to maximize the parallel processing power. For fully covered blocks, a simple scan filling algorithm is used. The half-space function is evaluated for every pixel for partially covered blocks.

Linear Block Rasterization

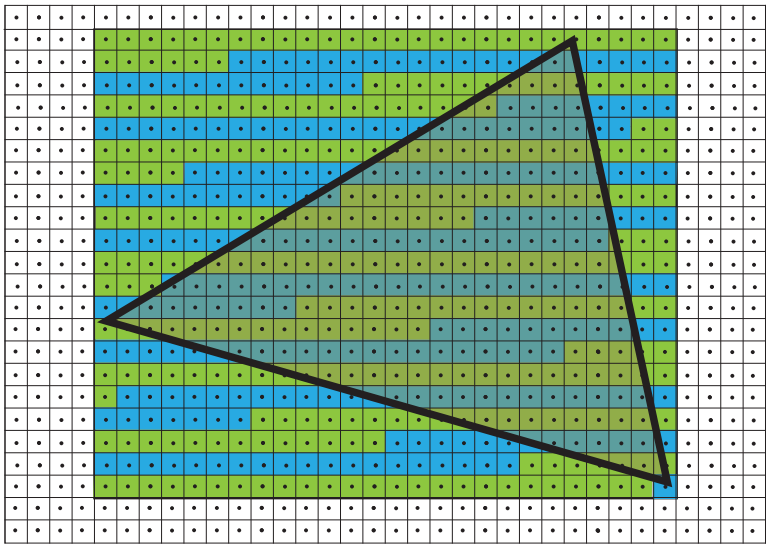


Figure 4.8: Linear Block Macro-Rasterization.

The Linear Block rasterization method is very similar to the '4x4 Block Half-Space Rasterization' method introduced in the previous section. Instead of evaluating the half-space function at the four corners of all blocks, this method does not require evaluation of any half-space functions in the first step. A triangle is subdivided into small linear blocks of 32 pixels as illustrated in Figure 4.8. The subdivided small blocks are appended to the global queue for post-processing. In the second step, a block is allocated with 32 threads (a warp size) sharing half-space function parameters that are pre-calculated in the first step. This technique is also based on a *divide and conquer* method, in order to allocate a thread per pixel.

Edge-Table Rasterization

The third method that is proposed is an Edge-Table Rasterization method. As explained earlier, the fundamental idea is the same, in that it subdivides a large polygon into small segments in order to allocate enough threads for each block. In the first step, this method scans a triangle from top to bottom and outputs an edge table as illustrated in Figure 4.9 [Pin88]. For each scan line, it finds the intersection of the scan line with the triangle such that it stores half-space parameters at the start and end position of each line to a global queue. In the second step, threads are allocated to each scan line segment to fill in all the pixels between parts of intersections.

The 4x4 Block Half-Space rasterization and Linear Block rasterization method achieve a similar performance, whereas the Edge-Table Rasterization method is about 20% slower, depending on the size of macro polygons. This is due to the variable length of each line segment.

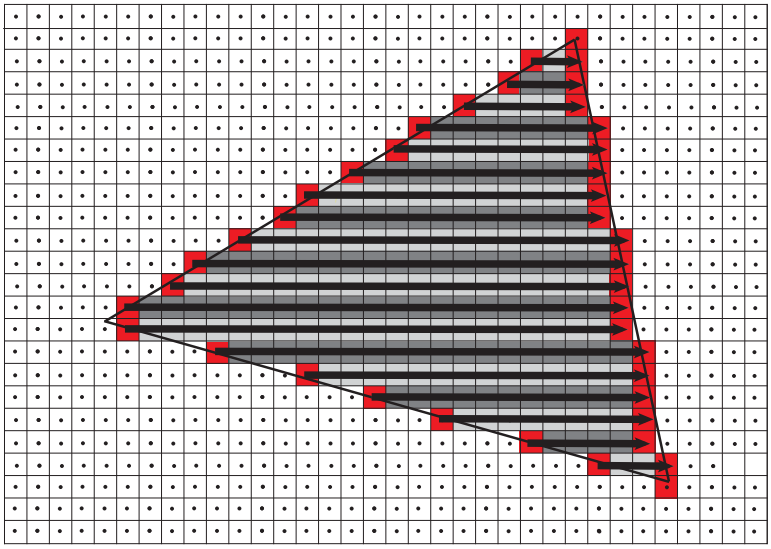


Figure 4.9: Edge-Table Macro-Rasterization.

4.5 Performance Analysis

This section shows the results of the proposed CUDA Deep rasterization method. The performance of our rasterization method is comparable with OpenGL and the latest GPU FreePipe [LHLW10, LHLW09a] method. All time measurements are taken on a commodity PC of Intel Core 2 Quad 2.66GHz with 4GB memory, and NVIDIA GeForce 480GTX with CUDA version 3.2.

Figure 4.10 shows the various models that are used in the performance analysis. Some models are chosen from Liu [LHLW10] in order to directly compare their results with ours timing as shown in Table **??**. A few other models are added in the test to examine the scalability of the methods for various situation. For instance, a simple cornell box (Figure 4.10.(e)) is the case where a few simple polygons occupy a large area in screen space (macro polygon) and the other extreme case is the Neptune objects' 4M micro polygons. The OpenGL fixed rasterization method is known to be the fastest hardware rendering method by using a dedicated rasterization module. However, in order to achieve this rate, a careful design is needed. For instance, for the 871K Dragon model, an *immediate mode* only offers 30 frames per second whereas a pre-compiled display list offers much higher frame rates. The Vertex Buffer Object(VBO) with redraw (glDrawElements) call reaches a maximum of 873 frames per second. The performance of the FreePipe method is directly measured from the running code in NVIDIA GeForce 480GTX. Note that these figures are higher than the ones presented in [LHLW10], because the graphics card is different. All performance timing does not include any color interpolation, normal interpolation, depth testing nor texture mapping. It is a simple single color rendering to buffers with only back-face culling enabled.

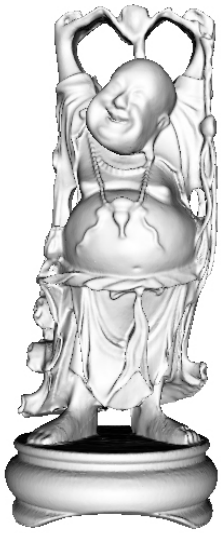
4.6 Discussion

In this chapter, a new CDUA deep rasterization method is presented, which allows building of Layer Hit Buffers in a thread efficient way. The results indicate that this CUDA rasterization is comparable to OpenGL and faster than FreePipe for most conditions. Spherical Layered Hit Buffers consist of a collection of LHBs in spherical directions, and this Deep Rasterization method could be further optimized to build a SLHB by adapting multi-view rasterization. In other words, this would involve triangle data being loaded once and rasterized in many orthogonal views at the same time by allocating efficient allocation threads. In this way, mesh data loading can be minimized.

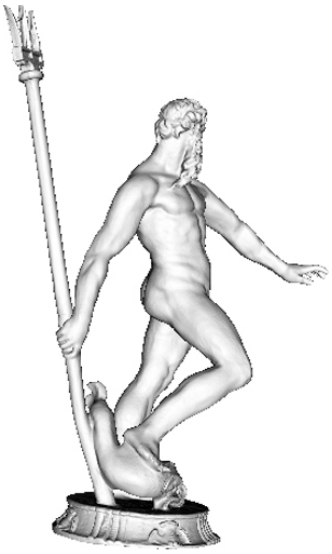
The latest development of OpenGL 4.0 and DirectX 11 provides a new feature, which enables a single-pass A-Buffer output in the hardware. Unlike the previous generation pipeline, this allows a list of fragments to be built per pixel during the rasterization by randomly write to the global memory. The new feature requires 5-20% additional cost to the conventional OpenGL rasterization method. Therefore our method provides better performance in most cases as described in Section 4.5.

(a) Bunny (70K triangles)

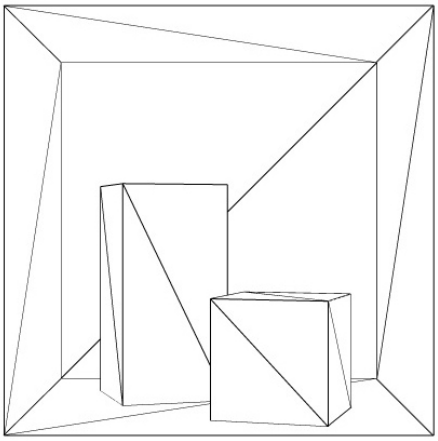
(b) Dragon (781K triangles)



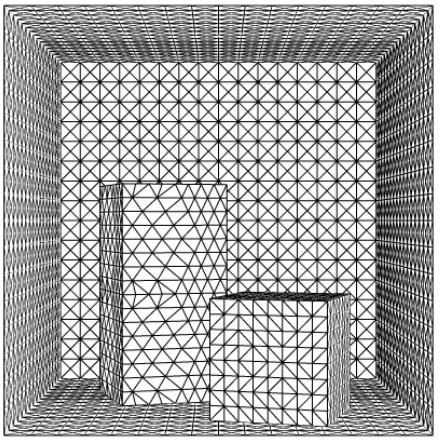
(c) Buddha (1M triangles)



(d) Neptune (4M triangles)



(e) Cornell (30 triangles)



(f) Tessellated Cornell (6.7K triangles)

Figure 4.10: Normal Rendering of Various Models.

Chapter 5

Real-Time GPU Global Illumination

In this chapter, a novel GPU-based global illumination method for real-time rendering of dynamic scenes is presented. In the previous chapter, a CUDA deep rasterization technique was discussed, which is able to build a visibility acceleration structure efficiently, using fast CUDA rasterization. The acceleration structure, based on Spherical Layered Hit Buffers(SLHB), is a set of directional Layered Hit Buffers(LHB) that represents a scene visibility field in a multi-layered structure. This chapter focuses on how to efficiently compute indirect lighting using this SLHB structure. A cone approximation technique is also introduced, in order to resolve the problem caused by a discrete directional representation of the SLHB.

5.1 Overview of the Rendering System

5.1.1 Rendering Procedure

This new rendering algorithm stems from a numerical Monte Carlo integration of the rendering equation. First of all, a scene is loaded and a bounding box is found, which encloses the scene and then rescales it to a unit cube. By applying this technique, it is ensured that that the scene fits a canonical LHB view for rasterization. A direction map is built in order to give an instant query of a random direction in a discrete directional representation of the SLHB. The number of directions needed for the SLHB is directly related to the accuracy of the acceleration structure.

Some possible directional sampling methods were introduced in Section 3.3. An icosahedron subdivision method is chosen to generate samples for the SLHB directions, and a Level 2 subdivision of an icosahedron generates 320 directions, which is an appropriate number of directions to represent approximated visibility in a scene. Another parameter that affects visibility accuracy is the size of the LHBs. The higher resolution means an improved accuracy, yet we opted to choose $128x128$ elements. Based on these figures, if we assume there are only on average 3 intersections per pixel in a LHB, the acceleration structure holds 63 million elements, containing depth and polygon information. This vast number of elements forms an approximate visibility field such that the instant visibility query is obtained at any position for any direction. The detail of how to construct a SLHB using a CUDA deep rasterizer was explained in the previous chapter, and once a SLHB for a static scene is built, the graphics hardware is used to build G-Buffers for camera viewing, which are color, normal and intersection buffers. Although

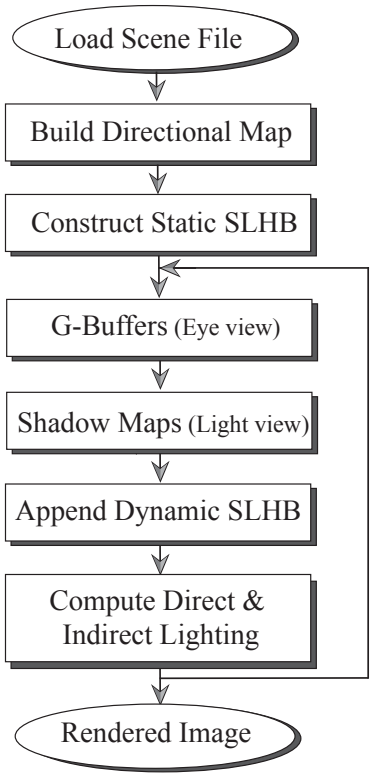


Figure 5.1: Overview of Rendering Procedure.

our proposed acceleration structure can be applied to area lights and Virtual Point Lights, we opt to use a point light source for a simplicity reason. A standard shadow map from a light view is built for the fast visibility test of direct illumination as illustrated in Figure 5.1. If there are dynamic objects in the scene, a second SLHB is constructed and appended to the static SLHB, in order to complete a full visibility field for both static and dynamic objects. In this way, a static SLHB is built only once, whereas a dynamic SLHB is updated for every frame. Although two SLHBs are maintained, the memory usage and access time is the same as for one big SLHB of the whole scene. Computation of direct and indirect illumination is described in Section 5.2 and Section 5.3.

5.1.2 Light Transport

This section describes how to organize the light transport and use the approximated visibility function in the equation. The new rendering solution follows the Neumann series expansion, and many global illumination algorithms are based on conventional shooting algorithms, such that the emitter propagates energy to all visibility surfaces in a scene. Although this approach works well in CPU oriented platforms, it is hard to implement on early GPUs due to a random write on multiple textures. In contrast, the new method is based on a gathering method at the screen space.

As stated in Section 3.2.3, the rendering equation can be rewritten in terms of light transport *Linear Operator* $\boldsymbol{T}$ as $L = L_e + \mathbf{T}L$, where L is the global illumination and L_e is the emitted energy from the light sources. Because of the unknown term, L, on both sides of the equation, a recursive expansion of the Neumann series is used to solve the equation. Our solution also follows the light energy transfer

through the Neumann series expansion, $L = L_e + \mathbf{T}L_e + \mathbf{T}^2L_e + \mathbf{T}^3L_e + ...$ where $\mathbf{T}L_e$ is direct lighting of emitters and $\mathbf{T}^2L_e$ is the first bounce of direct lighting and so on. In Section 3.2.2 it has been shown how to evaluate the transfer operator $\mathbf{T}$ in terms of numerical Monte-Carlo integration. Recalling equations 3.27, we replaced $\mathbf{T}L_e$ with L^1 and $\mathbf{T}^2L_e$ with L^2 as follows:

$$L(x \to \Theta) = \underbrace{L^0}_{\textbf{Emitter}} + \underbrace{L^1}_{\textbf{Direct illumination}} + \underbrace{L^2 \cdots L^n}_{\textbf{Indirect Illumination}} \tag{5.1}$$

$$= L_e(x \to \Theta) + \langle\, L_r(x \to \Theta) \,\rangle^A + \langle\, L_r(x \to \Theta) \,\rangle^H \cdots \tag{5.2}$$

The series expansion can be divided into three main groups. First, the emitter is denoted (L^0), secondly direct lighting is L^1 and finally the indirect illumination is $L^2 \cdots L^n$, which represents the second and multiple bounces of light as shown in Equation 5.1. The sum of these three terms gives the solution to the global illumination equation. Although the direct illumination can be computed in the same way as the indirect lighting, an important sampling of direct lighting gives better results with a fewer number of directions.

Compute_Radiance (x, Θ)

Initialize total radiance $L(x \to \Theta) = 0$

Compute Radiance at x in a direction Θ

$L(x \to \Theta) = L_e(x, \to \Theta)$ // Emitter (L^0)

$+=$ **Compute_Direct_Lighting ()** // Direct (L^1)

$+=$ **Compute_Indirect_Lighting ()** // Indirect $(L^2 \cdots L^n)$

Table 5.1: Compute Radiance in Neumann series.

The pseudocode of the light transport in terms of the Neumann series (Equation 5.1) is given in Table 5.1. In order to evaluate the light transport, N rays are generated at the eye point through the view plane (i, j); first to find the intersection position x. At this intersection position, we evaluate the radiance $L(x \to \Theta)$ calling the **Compute_Radiance** function in Table 5.1, where Θ is a direction at x toward the eye. A hardware rasterized G-Buffer method is employed to evaluate the primary intersection (x) and a ray direction (Θ).

The pseudocode for the direct lighting function **Compute_Direct_Lighting** is given in Table 5.2. Computation of direct lighting is achieved by applying the *Area formulation* operator $\langle\, L_r(x \to \Theta) \,\rangle^A$ in Section 5.2, which explicitly samples the light sources. In the same way as other real-time global illumination solutions, the hardware is explicitly used to generate a shadow map to provide an occlusion query for direct illumination computation.

Finally, Table 5.3 describes the **Compute_Lighting** function to evaluate the indirect illumination. The indirect light term, $L^2 \cdots L^n$, is the most expensive operation in global illumination and the *Hemisphere formulation* operator $\langle\, L_r(x \to \Theta) \,\rangle^H$ is applied. In order to quickly find the visibility in random directions, the proposed SLHB structure is employed for the occlusion query of all indirect lighting computing. The detailed description of how to employ the SLHB is explained in Section 5.3.

Compute_Direct_Lighting ()

For N shadow rays

// Generate random point on the light source

Generate random numbers $\xi_1, \xi_2 \in (0,1)$

Compute a point y_i at the light source with PDF $p(y_i)$ using ξ_1, ξ_2

Evaluate $V(x, y_i)$ from the shadow map

Apply 'Area Formulation' with a local BRDF operator

$$\langle L_r(x \to \Theta) \rangle^A = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{L_e(y_i \to \overrightarrow{y_i x}) f_r(x, \overrightarrow{xy_i} \to \Theta) V(x, y_i) G(x, y_i)}{p(y_i)} \right]$$

Table 5.2: Direct Lighting.

Compute_Indirect_Lighting ()

For N indirect samples

// Generate random directions on hemisphere

Generate random numbers $\xi_1, \xi_2 \in (0,1)$

Compute a direction Ψ_i with PDF $p(\Psi_i)$ using ξ_1, ξ_2

Evaluate $V(x, \Psi_i)$ from the SLHB acceleration structure

Apply 'Hemisphere formulation' with a local BRDF operator

$$\langle L_r(x \to \Theta) \rangle^H = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{L(x \leftarrow \Psi_i) f_r(x, \Psi_i \to \Theta) V(x, \Psi_i) \cos(N_x, \Psi_i)}{p(\Psi_i)} \right]$$

Compute $L(x \leftarrow \Psi_i)$ recursively by calling **Compute_Radiance** $(x, -\Psi_i)$

Table 5.3: Indirect Lighting.

5.2 Direct Lighting

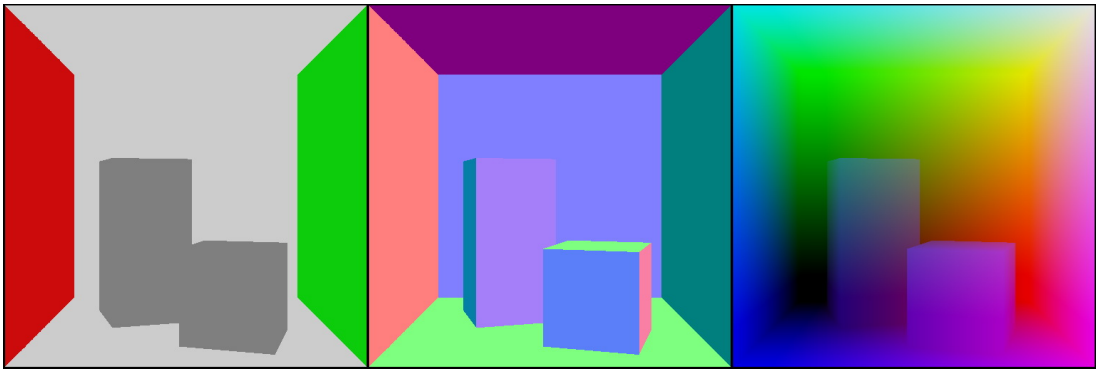
The direct illumination can be computed with either *Area formulation* or *Hemisphere formulation* methods. A straightforward method for solving the direct illumination using hemisphere formulation is to sample the directions over the hemisphere and disregard the rays, which do not intersect with light sources. However, a numerical *Area formulation* method is used here, as given in Equation 5.3, since there is prior knowledge of where the light are in the scene, such that an importance sampling scheme can be adapted to take advantage of the initial lighting environment.

$$\langle L_r(x \to \Theta) \rangle^A = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{L_e(y_i \to \overrightarrow{y_i x}) f_r(x, \overrightarrow{xy_i} \to \Theta) V(x, y_i) G(x, y_i)}{p(y_i)} \right] \tag{5.3}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \left[\underbrace{\left[\underbrace{\left(\frac{L_e(y_i \to \overrightarrow{y_i x}) V(x, y_i)}{p(y_i)} \right)}_{\text{Ray Casting}} \underbrace{G(x, y_i)}_{\text{Form Factor}} \right]}_{\text{Transport}} \underbrace{[f_r(x, \overrightarrow{xy_i} \to \Theta)]}_{\text{Scattering}} \right] \tag{5.4}$$

The equation can be grouped into two parts, one for the *transport operator* and the other for the

scattering operator. The scattering operator is a four dimensional function that defines how light interacts with a surface, known as the BRDF (Bidirectional Reflectance Distribution Function). The transport operator includes *Ray Casting* and *Form Factor* computation. A visibility function $V(x, y_i)$ has been explicitly presented in Equation 5.4 to show where the visibility function is applied in *Ray Casting*. Conventionally, the ray casting method has been used to determine the visibility $V(x, y_i)$; however, many real-time algorithms tend to use graphics hardware to obtain the visibility for primary rays, since it is capable of full determination of primary intersection positions using a simple rasterization method. We also use a standard shadow map technique to find the visibility in direct illumination. A shadow map is created by rendering a scene from the light point of view, where $V(x, y_i)$ can be easily obtained by simply comparing the depth values. To improve the accuracy of the visibility function, the resolution of the shadow map can be adjusted. Additionally, an omnidirectional shadow map or a paraboloid shadow map method offers a wider field of view for shadow maps. Our rendering solution explicitly uses the shadow map instead of ray casting methods to evaluate direct lighting illumination. The proposed CUDA deep rasterization method can therefore be modified to build a shadow map. Since CUDA deep rasterizers build a list of depth information per pixel, the *atomicMin* function can be used to find minimum depth values for each pixel. There are many techniques to improve the quality of shadow, one of the more popular methods is called Percentage Close Filtering (PCF), which performs a filtering on the shadows by using multiple samples from the shadow map.



(a) Color (b) Normal (c) Position

Figure 5.2: An Example of a G-Buffer.

Direct illumination computing occurs at the intersection positions where the eye first hits the objects in a scene. In other words, it occurs where the primary rays intersect with the scene. Graphics hardware is used to build G-buffers, which contain the color, normal and intersection positions in buffers as shown in Figure 5.2. In the same way, it is much faster to compute primary ray intersections by using a hardware rasterization method.

5.2.1 Important Sampling on Luminaries

In order to evaluate Equation 5.4, two more variables y_i and $p(y_i)$ should be defined. y_i denotes the sampled points on the emitters and $p(y_i)$ represents the probability density function (PDF). As a numerical

Monte Carlo integration method is used for direct lighting, first the random sample points are generated and the radiance values are averaged over N samples to approximate illumination. In the case of rectangular luminaries, uniform random variables ξ_1, ξ_2 in the range of 0 and 1 are chosen. A corner vertex, x_0 and two vectors $\overrightarrow{v1}$ and $\overrightarrow{v2}$ form a plane to generate a uniform random sample (y_i):

$$y = x_0 + \xi_1 \overrightarrow{v_1} + \xi_2 \overrightarrow{v_2} \tag{5.5}$$

This sampling function has a constant density function ($p(y_i)$) value of:

$$PDF = p(y) = \frac{1}{\| \overrightarrow{v_1} \times \overrightarrow{v_2} \|} \tag{5.6}$$

This simple sampling function is used to generate uniform random points, y_i, on a rectangular emitter with PDF, $p(y_i)$. Therefore, Equation 5.4 can be evaluated with intersection point x, normal Nx (obtained from G-buffers) and visibility information $V(x, y_i)$ from the shadow map, together with random point y_i and PDF $p(y_i)$. The sampling functions of other shapes of luminaries are discussed in [SWZ96].

5.3 Indirect Lighting and Irradiance Estimation

The solution of the global illumination problem is equivalent to solving the *rendering equation*. However, the unknown radiance function L appears on both sides of the equation in 3.6. Thus it is difficult to solve in an analytical manner. For this reason, Monte Carlo numerical approximation is employed to resolve the integral function. In the previous section, *Area formulation* is chosen to compute the direct lighting, whereas *Hemisphere formulation* can approximate the integral function better for indirect illumination, as given in Equation 5.7.

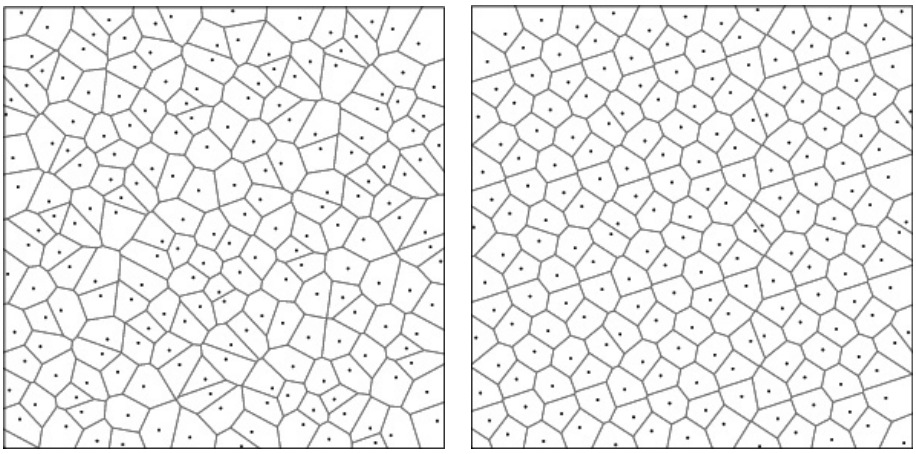
$$\langle L_r(x \to \Theta) \rangle^H = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{L(x \leftarrow \Psi_i) f_r(x, \Psi_i \to \Theta) \cos(N_x, \Psi_i)}{p(\Psi_i)} \right] \tag{5.7}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \underbrace{\left[\underbrace{\left[\underbrace{L(x \leftarrow \Psi_i) V(x, \Psi_i)}_{Raycasting} \frac{\cos(N_x, \Psi_i)}{p(\Psi_i)} \right]}_{Transport} \underbrace{[f_r(x, \Psi_i \to \Theta)]}_{Scattering} \right]} \tag{5.8}$$

Again, the visibility function $V(x, \Psi_i)$ is explicitly included in the *ray casting* function as a part of the transport function in Equation 5.8. The visibility function $V(x, \Psi_i)$ represents occlusion at x from all incoming directions (Ψ_i). To compute this, a fast ray casting method is necessary. The most computationally expensive part in global illumination is the visibility computation, and many techniques have been developed to accelerate the operation, such as hierarchical bounding volumes, Octrees and BSP trees. In this thesis it is proposed to use Spherical Layered Hit Buffers as an acceleration structure to find occlusion information instantly. In this section, a hemisphere integration with SLHBs for indirect illumination is presented.

Generating Low Dependency Indirect Samples (Ψ_i)

When the eye ray hits an object, the direct lighting is computed as described in Section 5.2. At the intersection point, indirect lightning is computed by evaluating the numerical integral equation 5.8. To



(a) Halton Samples (b) Hammersley Samples

Figure 5.3: Low Dependency Sampling (196 samples).

compute the equation, random directions $\Psi(\theta,\phi)$ must first be defined. Since the integral domain is a hemisphere, evenly distributed points should be generated over this, and a pair of two random variables ξ_1,ξ_2 forms unbiased samples in a square region. By projecting these variables into spherical coordinates, uniform random samples in the hemisphere are obtained using the transform equation 5.9.

$$\Psi(\theta,\phi) = (2\arccos(\sqrt{1-\xi_1}), 2\pi\xi_2) \tag{5.9}$$

The random directions are defined in cartesian coordinates as:

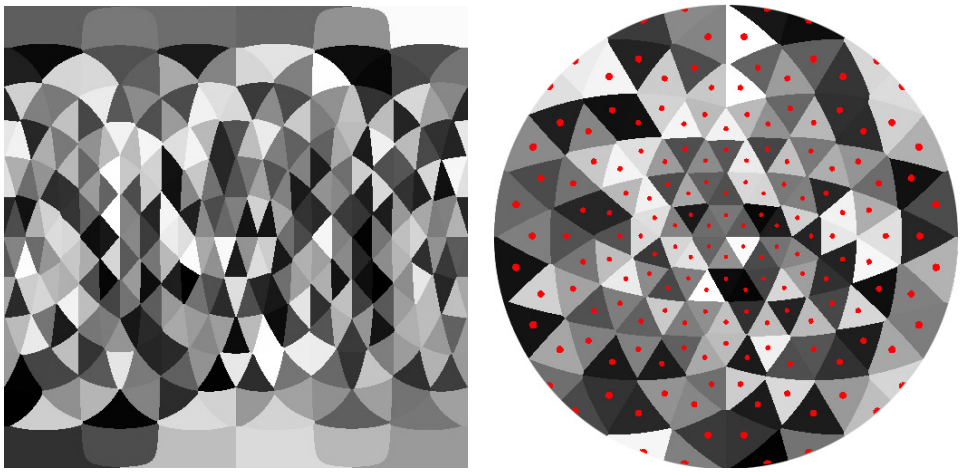
$$\Psi = \begin{array}{rcc} x & = & \cos(2\pi\xi_1)\sqrt{1-{\xi_2}^2} \\ y & = & \xi_2 \\ z & = & \sin(2\pi\xi_1)\sqrt{1-{\xi_2}^2} \end{array} \tag{5.10}$$

The PDF function on this distribution is just a constant value over the surface area of a hemisphere. Therefore the PDF is $1/2\pi$. The uniform random variables ξ_1,ξ_2 result in high variance and slow convergence, so in order to lower the variance, the stratified sampling technique is commonly used. However, in this research we employ a low-discrepancy sequence, also called a quasi-random sequence. Figure 5.3 shows an example of two low-discrepancy sampling methods; the Halton sampling and Hammersley techniques. Although it is possible to compute low-discrepancy samples on the fly, we opt to pre-compute the multi-set of the Halton sequence, which is stored in the constant memory in order to minimize computation. Two random variables are used to determine the set number, and the other variable is used to select the pre-defined sample in the sequences. By adapting the Halton sequences, Monte Carlo integration is more accurate with fewer samples. However, the more samples that are used, the better the approximation that is achieved.

Directional Map

Once a random indirect direction Ψ_i is chosen, it is placed into the SLHB parameterization in order to find the corresponding LHB direction. However, finding the closest direction in a discrete SLHB

parameterization is not a simple task. For this reason, we propose a directional map, which consists of an indexed texture to quickly find the nearest direction.



(a) Longitude-Latitude Map (b) Paraboloid Map

Figure 5.4: A Directional Map of an icosahedron (320) is illustrated with Longitude-Latitude and Paraboloid parameterizations.

Figure 5.4 represents an icosahedron directional map of the SLHB. The left-hand image uses an indexed color to represent the distribution of directions in a longitude-latitude map. Each triangular area represents a single direction of the LHB(θ,ϕ). The right-hand image shows the center of each direction in red dots, in a paraboloid representation. The random direction Ψ_i is projected on to the directional map, then the nearest direction of the LHBs is chosen by finding the index i at the projected position in the longitude-latitude map. In this way, a constant query to find the nearest LHB direction (LHB_i) for a given direction Ψ_i is achieved. The directional map serves as a look-up table for the instant directional query, which is pre-computed and stored in a texture map.

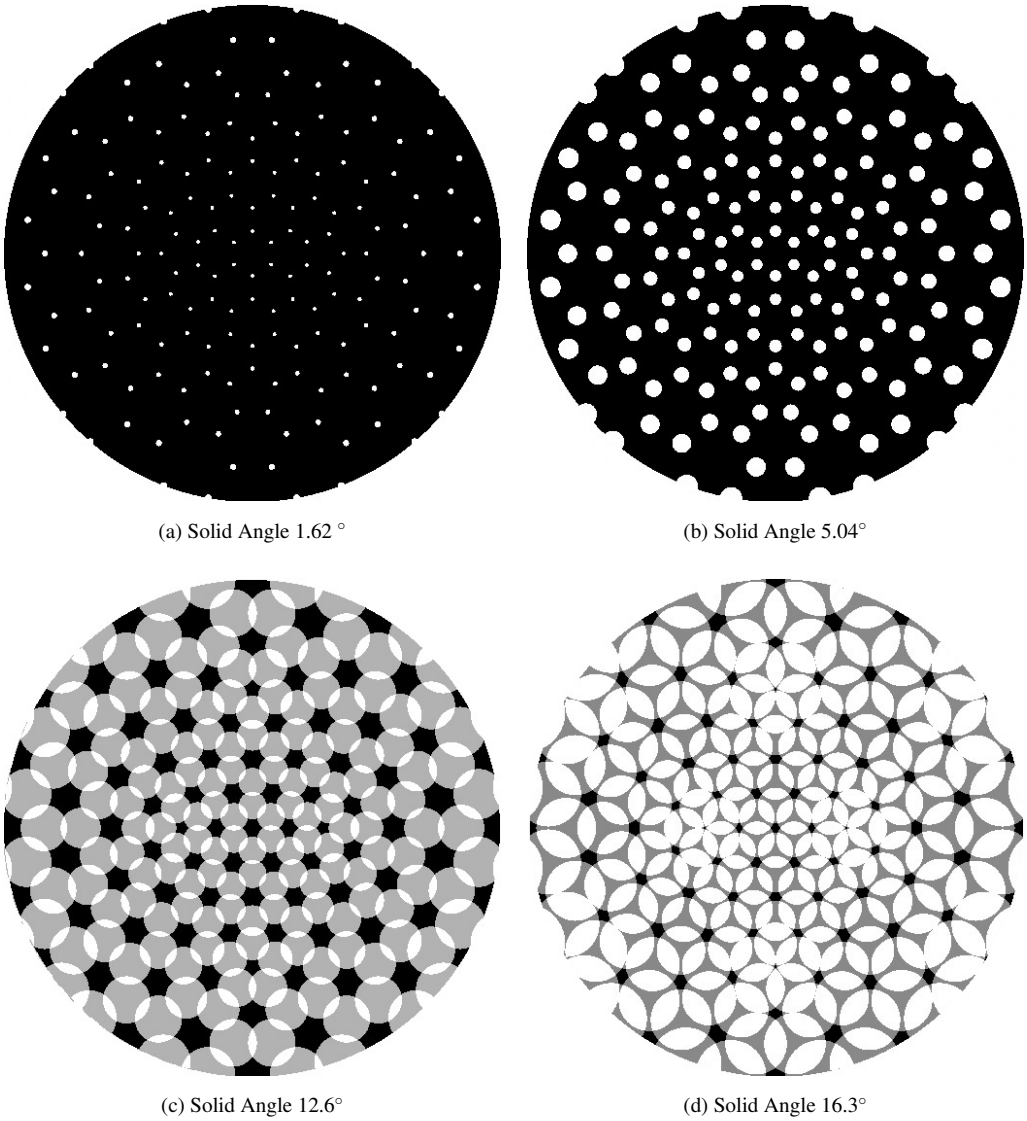
Irradiance Estimation

The hemisphere formulation of indirect illumination in Equation 5.8 can be numerically solved when a visibility function $V(x,\Psi_i)$ and incoming radiance function $L(x \leftarrow \Psi_i)$ of the *ray casting* term are given. One of the most expensive parts of indirect illumination is to evaluate the visibility function. We propose the SLHB acceleration structure to solve the ray casting term efficiently. The visibility term, $V(x,\Psi_i)$ finds the nearest intersected object from the ray origin $\mathbf{x}$ in direction Ψ_i.

When a random direction Ψ_i is given, its corresponding direction (LHB_i) is found by looking up the directional map. The ray origin $\mathbf{x}$ is then projected onto the SLHB space to find the nearest object in the LHB_i direction. In other words, the ray origin $\mathbf{x}$ in world space is multiplied by $MVP(\theta,\phi)$ in Equation 4.1, which gives a new projected 3D location in the LHB_i direction, where the projected position's x,y values represent the 2D pixel location in an LHB buffer ($LHB_i(x,y)$), and the z value gives the depth value in along the ray direction LHB_i. To find the nearest object, the projected z value is compared with depth values in the element list at $LHB_i(x,y)$. The element with the smallest depth difference value

is the nearest intersected polygon. In this way, the visibility function $V(x,\Psi_i)$ can be used to find the nearest polygon ID. The depth value of the nearest polygon along the Ψ_i direction is used to derive the intersection position in world space by applying the inverse $MVP(\theta,\phi)$. It is straightforward to find the incoming radiance function $L(x \leftarrow \Psi_i)$ in Equation5.8 when this intersection is obtained. It is evaluated by recursively calling the **Compute_Radiance** function with new parameters. Therefore, Equation 5.8 can be solved efficiently using the proposed SLHB acceleration structure.

5.3.1 Cone Approximation



(a) Solid Angle 1.62 ° (b) Solid Angle 5.04°

(c) Solid Angle 12.6° (d) Solid Angle 16.3°

Figure 5.5: Solid angles coverage shown in Paraboloid Map (Icosahedron 320 directions) (a) closest to a delta function (c) solid angle covers the hemisphere region.

Solid Angle Estimation

The hemisphere formulation of indirect illumination can be efficiently computed using a SLHB structure. However, the SLHB is a set of discrete LHB directions (LHB_N) to represents the continuous directions

LHB(θ,ϕ). Therefore, a SLHB has a limited number of directions as shown in Figure 5.4. In order to visualize the directions in terms of solid angle, an example with 320 icosahedron directions is illustrated in Figure 5.5. The red dots in Figure5.4 are located in the same position in the solid angle maps in Figure 5.5(a).

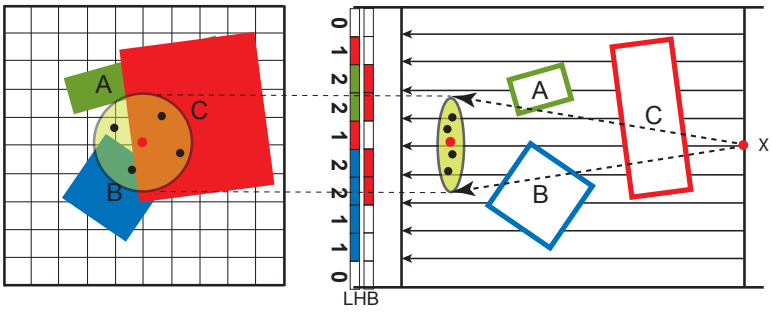
As shown in Figure 5.5(a), a set of discrete directions (LHB_N) can only represent delta functions, which means that only exactly matched directions can be retrieved from the SLHB. In other words, whenever a random direction $\Psi(\theta,\phi)$ is queried, only the nearest direction LHB_i is given by checking the directional map. In order to avoid the discretization problem, each direction in the SLHB space should cover a solid angular coverage instead of the delta function. Figure 5.5 shows the various solid angle coverages in a paraboloid map. If the discrete directions (or delta functions) are extended to cover 12.6 $^\circ$ of solid angle as shown in Figure 5.5(c), then they overlap each other such that a better coverage of the hemisphere can be achieved from the discrete directions. Figure 5.5(d) shows that the 320 discrete directions with a 16.3° solid angle can give full coverage of the hemisphere area. Therefore, an estimated solid angle coverage can be obtained for any parameterization of hemispherical representation discussed in Section 3.3.

Cone Approximation

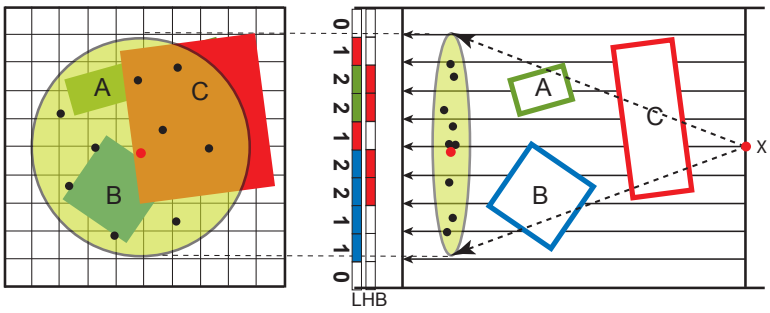
The irradiance estimation in Section 5.3 can be extended to support the solid angle estimation. A *Cone Approximation* is proposed to simulate a solid angle coverage for the discrete representation of the SLHB. Since the integration domain is on a hemisphere in Equation 5.8, each discrete LHB direction should cover a solid angular region. The cone shape approximation for a solid angle on the hemisphere is similar to a disc approximation in the LHB representation.

The *ray casting* term involves finding the nearest object towards a random direction Ψ_i at the ray origin $\mathbf{x}$ in both a visibility function $V(x,\Psi_i)$ and incoming radiance function $L(x \leftarrow \Psi_i)$. When the ray origin $\mathbf{x}$ is projected into $LHB_i(x,y)$ location in a LHB buffer, the Cone Approximation can be evaluated by jittering the projected position (x,y) within the solid angle disc region. Therefore, the new location is at $LHB_i(x+\xi_1, y+\xi_2)$, where ξ_1 and ξ_2 are random variables within the disc region. In short, a random direction $\Psi(\theta,\phi)$ is achieved by firstly selecting a random direction LHB_i and then choosing two random variables ξ_1 and ξ_2 to jitter the projected location $LHB_i(x,y)$ to $LHB_i(x+\xi_1, y+\xi_2)$. The nearest object at $LHB_i(x+\xi_1, y+\xi_2)$ is obtained in the same way in the irradiance estimation in Section 5.3.

Figure 5.6 shows two cone approximations with $r = 1.5$ and $r = 4$ pixels. The right-hand column is a canonical LHB view and the left-hand column is in a 2D LHB buffer view. The red dots represent the delta direction at x and then the projected z value should be compared with a list of depth values at the red dot position in order to find the nearest polygon, which in this case is C . However, the cone approximation jitters the red dots within the solid angle coverage (which is the yellow disc in the LHB) and find the nearest polygon at a jittered location. As an example, Figure 5.6 (a) shows 4 randomly placed samples on the disc in a black color. Two samples intersect with the C polygon and one intersects with the B polygon. A wider solid angle coverage ($r = 4$) with more samples can give a better estimation of the numerical Monte Carlo integration over the hemisphere. A low dependency sampling scheme



(a) R=1.5 pixel



(b) R=4 pixels

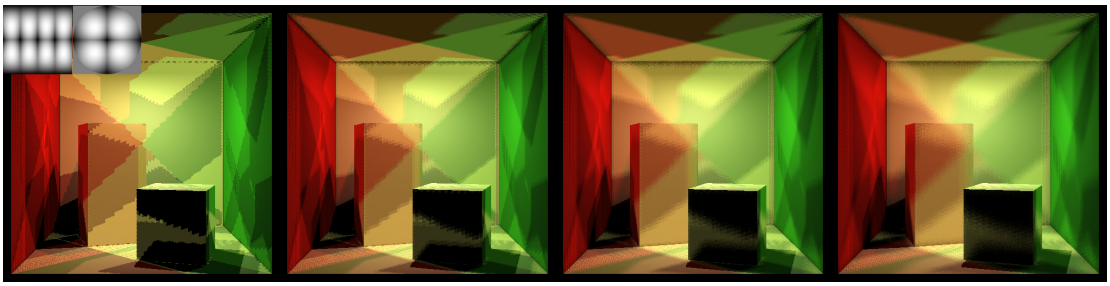
Figure 5.6: An example of two Cone Approximations (R=1.5, 4 pixels) in a canonical LHB view.

such as the Halton sequence can be adapted to place samples on the disc to provide a fast approximation with a small number of samples.

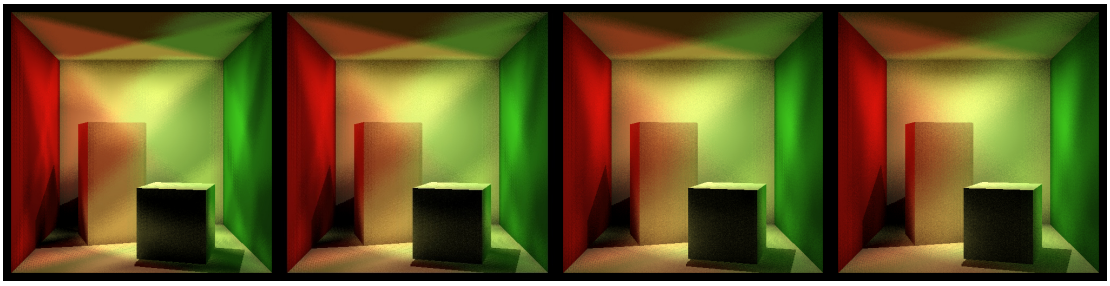
Figure 5.7 (a) shows an extreme case, where the number of LHB directions is limited to only 8 directions, which is called an octahedron distribution. When the number of LHB directions is low, the solid angle coverage for each direction becomes too wide. Although it takes a small amount of time to build a small SLHB, the rendered image suffers severely from banding artifacts as shown in Figure 5.7 (a). This example illustrates that the cone approximation technique can remove the high-frequency structural noise by increasing the radius of the disc (or solid angle) while maintaining the same number of irradiance samples. The first image ($r = 1$ pixel) shows that the exact (or delta) directions result in severe artifacts due to a very limited number of directions. The last image shows that a large disc estimation can resolve these artifacts. In this example, an octahedron distribution with 8 directions requires at least a 40° solid angle, which is around $r = 100$ in the cone approximation, to estimate irradiance distribution over the hemisphere accurately.

5.3.2 Lambertian Reflection Model

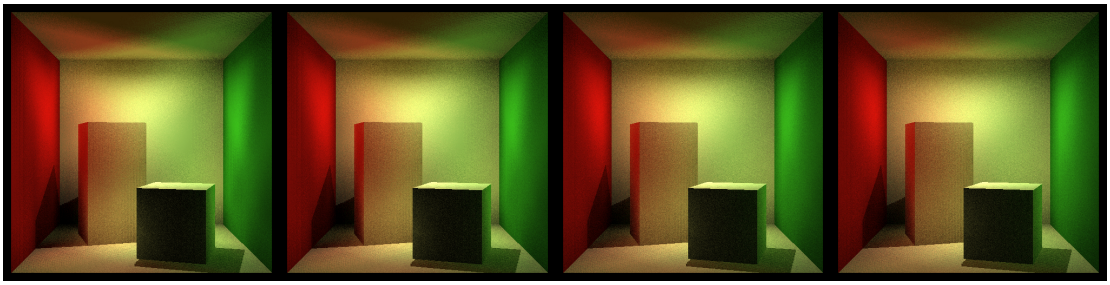
An example of the Lambertian reflection model is introduced in this section. By carefully choosing direction samples Ψ_i, it is possible to cancel out the cosine term and the PDF function. In the case of the Lambertian reflection model, cosine weighted directions Ψ are chosen to simplify the integral function. The random variables ξ_1, ξ_2 are uniformly sampled in canonical space between 0 and 1. Two random



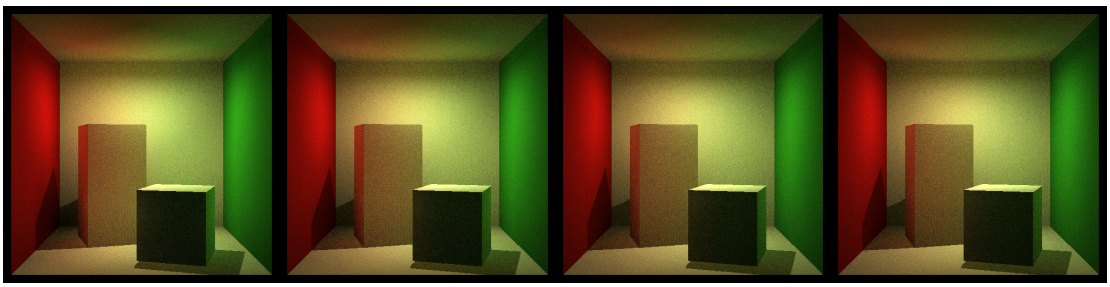
(a) $r = 1$ (b) $r = 4$ (c) $r = 7$ (d) $r = 10$



(e) $r = 16$ (f) $r = 21$ (g) $r = 28$ (h) $r = 33$



(i) $r = 40$ (j) $r = 45$ (k) $r = 54$ (l) $r = 60$



(m) $r = 72$ (n) $r = 84$ (o) $r = 96$ (p) $r = 100$

Figure 5.7: A Cone Approximation for 8 LHB directions with various radius r (in pixels). The number of irradiance samples is the same for all cases.

variables form a direction Ψ on the upper hemisphere of the xz plane:

$$\Psi = \begin{array}{rcl} x & = & \cos(2\pi\xi_1)\sqrt{1-\xi_2} \\ y & = & \sqrt{\xi_2} \\ z & = & \sin(2\pi\xi_1)\sqrt{1-\xi_2} \end{array} \tag{5.11}$$

Then the Probability Distribution Function (PDF) becomes a cosine over pi.

$$PDF = p(\Psi) = \frac{\cos(N_x, \Psi)}{\pi}$$

The BRDF of the Lambertian model is constant:

$$BRDF = f_r(x, \Psi \leftrightarrow \Theta) = K_d = \frac{\rho_d}{\pi}$$

The notation of the numerical solution to the integral function for a surface is:

$$L(x \to \Theta) = L_e(x \to \Theta) + L_r(x \to \Theta) \tag{5.12}$$

$$\simeq L_e(x \to \Theta) + \langle\, L_r(x \to \Theta) \,\rangle^H \tag{5.13}$$

The estimation of the reflected radiances then becomes very simple. When a hemisphere formulation is used to evaluate the Monte Carlo Integral, it becomes:

$$\langle\, L_r(x \to \Theta) \,\rangle^H = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{L(x \leftarrow \Psi_i)\, f_r(x, \Psi_i \to \Theta) \cos(N_x, \Psi_i)}{p(\Psi_i)} \right] \tag{5.14}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \left[\frac{L(x \leftarrow \Psi_i)\, \frac{\rho_d}{\pi} \cos(N_x, \Psi_i)}{\frac{\cos(N_x, \Psi_i)}{\pi}} \right] \tag{5.15}$$

$$= \frac{\rho_d}{N} \left(\sum_{i=1}^{N} L(x \leftarrow \Psi_i) \right) \tag{5.16}$$

This equation describes the result that the reflected radiance energy on a surface is a simple procedure of accumulating the irradiance energy from visible surfaces in the direction Ψ_i. Therefore, the radiance equation is:

$$L(x \to \Theta) \simeq L_e(x \to \Theta) + \frac{\rho_d}{N} \left(\sum_{i=1}^{N} L(x \leftarrow \Psi_i) \right) \tag{5.17}$$

This is extremely fast computation since $L(x \leftarrow \Psi_i)$ needs to find the nearest intersection point towards the direction $-\Psi_i$ at x. The SLHB structure offers an instant visibility query and gives $L(x \leftarrow \Psi_i)$ very efficiently.

5.3.3 Implementation Details

The CUDA architectures allows the user to manipulate *constant memory* and *shared memory* as a user defined cache. These memories have no latency, and function in the same way as registers in core processors. Therefore, it is important to allocate the most frequent variables in the constant memory, which are read-only variables. In these algorithms, matrices such as the Light viewing transform, Camera viewing transform and LHB viewing transform are left out. A set of pre-computed Halton samples are also stored in the constant memory. The shared memory is used extensively in the rendering algorithm and macro rasterization, and the rest of the data, such as the SLHB and rand seeds are stored in *global memory* in the GPU.

Random Number Generation

In order to evaluate Monte Carlo path tracing, an efficient random number generator is essential. One of the difficulties in implementing a quasi-random number generator in CUDA is the serialization problem.

A seed number is called from many threads concurrently, which causes a thread waiting problem. To resolve this issue, we use a random seed in the size of viewport such that individual seed numbers are updated whenever the random function is called. Langdon's [Lan09] random number generator is used, which offers a fast, high quality random number generator in CUDA. Our proposed rendering algorithm naturally supports a Monte Carlo path tracing method, which computes the radiance for each pixel by randomly sampling paths and computing the radiance along those paths.

5.4 Dynamic Elements

The proposed rendering method can realize real-time walk-through of complex virtual environments, computing multiple indirect bounces on the fly. A full analysis of our rendering method is examined in Chapter 6. There are many parameters that alter the quality and rendering speed of the proposed solution; for instance the number of SLHB directions, the resolution of the LHB, the number of samples for shadow, direct and indirect illumination and the number of bounces for indirect lighting. The results indicate this solution is also capable of simulating fully dynamic environments, including changes of view, materials, lighting and objects at interactive rates on a commodity graphics hardware. It supports not only a point light source, but also complex area lighting.

Figure 5.8: A galloping dynamic horse (17K) running at 45 frames per second at 512 x 512.

Figure 5.8 shows an example of a galloping horse, running at over 40 frames per second, with a few bounces of indirect illumination. The performance analysis of dynamic objects is discussed in Section 6.4. The proposed rendering algorithm is capable of rendering a global illumination scene in real-time, which includes rebuilding the acceleration structure for each frame. It is possible to build the SLHB in real-time for large models, but a method is proposed here that uses two SLHBs and merges them on the fly, since the acceleration structure is very flexible. This means the new method is capable of dealing with dynamic objects. To do this, a SLHB for the static part of a scene is built only once, whereas the dynamic objects are appended to the static SLHB on the fly for every frame. In this way, more resources

are allocated to computation of the illumination rather than to building acceleration structures. A LHB consists of counters and buffers, which hold a list of elements. By maintaining two counters, one for static and the other for dynamic information, a new list of elements from dynamic objects is appended to the end of the static element list, which increments the counters. Therefore, only the static SLHB is not overwritten by the dynamic SLHB. More details and performance analyses are given in Section 6.4.1

5.5 Discussion

The proposed rendering algorithm can be extended to support an environmental map lighting for real-time rates without any pre-processing step. Figure 5.9 shows an example of the GPU path-traced environmental lighting for the same model, under different lighting conditions.

Environment maps are 2D textures that store the directional radiance arriving at a single point. It is assumed that the environmental light is infinitely far away, and a simulation of environmental lighting requires an integral of all incident lights multiplied by the BRDF. The diffuse materials require many samples over the hemispheres to accurately simulate the integration function. Using *hemisphere formulation* together with the SHLB acceleration structure means that real-time rendering of an environmental lighting condition can be achieved.

Figure 5.9: An Example of Environment Map Lighting.

In this chapter, the light transport has been analyzed in terms of approximated visibility. A GPU-based real-time rendering method has been presented to solve the integral equation in numeral approximation, using Monte Carlo integration. An approximated visibility structure, SLHB, is presented to provide instant occlusion query for indirect illumination. It has been shown that the proposed GPU rendering algorithm is cable of supporting fully dynamic environments at real-time rates. In the following chapter, an in-depth analysis of the proposed Spherical Layered Hit Buffers and real-time rendering

solutions will be performed.

Chapter 6

Results

In this chapter, the results of the proposed algorithm presented in previous chapters is discussed. Firstly, the scalability of the algorithm is examined on a number of polygons, in a number of directions and with various sizes of hit buffers. Then a rendering performance is presented on several complex static objects, which is then compared with a recent path tracing technique provided by GPU Optix SDK. Finally, this chapter demonstrates the performance of the algorithm on dynamic objects, followed by a scalability test on various graphics cards.

All the results reported here are obtained using an Intel Core 2 Quad processor at 2.66GHz, with 4GB of memory. The application has been implemented on modern graphics hardware using OpenGL and CUDA 3.2. The graphics card used in these results is an NVIDIA Geforce GTX 480 card, with 1.5GB RAM. Recent OpenGL extensions are also used, for example floating-point textures, frame buffer objects and pixel buffer objects. All codes was written in C++ and complied on Visual Studio .NET running on Windows XP. Due to some limitation of the CPU time functions in measuring parallel processing, the CUDA based time functions are used to obtain the accurate time. Most time measurements are presented in milliseconds (ms). All images shown in this results chapter are rendered at a screen resolution of 512x512, except Figure 6.7, 6.8 and 6.9.

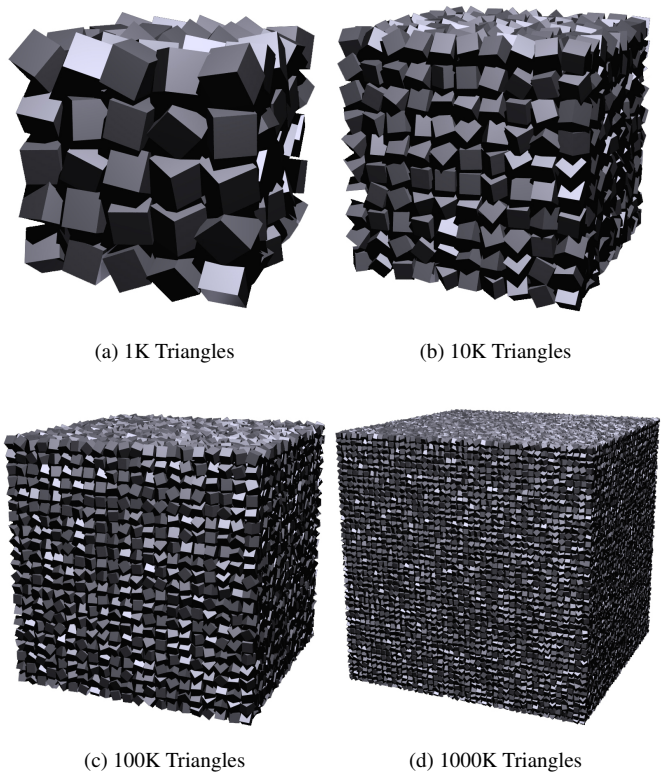
6.1 Evaluation of the Spherical Layered Hit Buffer

In this section the performance of the GPU-based Monte-Carlo path tracing method is evaluated. There are two parameters which have the most influence on the performance of the Spherical Layered Hit Buffers (SLHB). Initially, the relationship between increasing the number of sampling directions and the size of the Hit Buffers is evaluated. Scalability data is gathered by varying the number of LHB directions, the Hit Buffer size and the number of polygons. A measurement of time is recorded in order to build the acceleration structure and render the scene, with a size of 512 x 512 accounting for 64 paths per pixel, with 2 bounces of indirect illumination computation.

6.1.1 Test Scenes for Scalability

Two sets of test scenes are selected specifically in order to measure the various conditions for the SLHB scalability test. The first one is a collection of random cubes in regular grids as shown in Figure 6.1. To generate the scenes, a number of unit cubes are allocated to axis-aligned locations and then a random

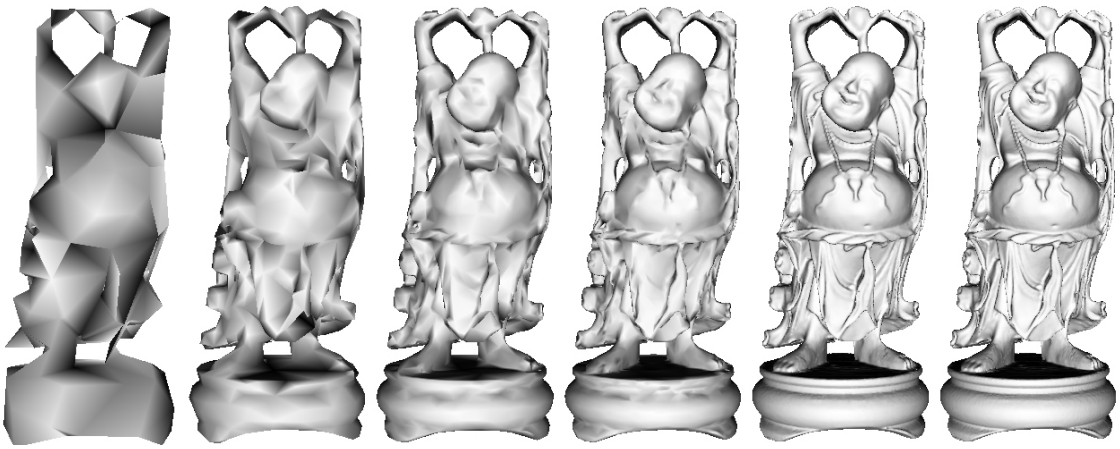
rotation function is applied to each cube to avoid perfect alignment. Test scenes have been generated in the range from one thousand to one million polygons.



(a) 1K Triangles (b) 10K Triangles

(c) 100K Triangles (d) 1000K Triangles

Figure 6.1: Random unit box objects for scalability test.



(a) 2K Triangles (b) 4K Triangles (c) 8K Triangles (d) 16K Triangles (e) 250K Triangles (f) 1000K Triangles

Figure 6.2: Multiresolution Happy Buddha objects for scalability test.

The second test scene is taken from the Stanford 3D Scanning Repository [Std04]. A Happy Buddha model is a complex object that is widely used for test purposes, which originally consisted of one million triangles. In this research, a commercial software package is used to generate several multi-resolution

objects of a similar size to ensure the same level of polygon occupancy in the SLHB for all objects. However, the detail of the Buddha model is in various levels as illustrated in Figure 6.2. The memory requirement to build the SLHB is in the range of 15MB to 480MB for the various multi-resolution objects. The initial results show that the performance of the algorithm on both the scenes is very close, so the time measurement on the Buddha model for the scalability test is only shown in the following sections:

6.1.2 Numbers of Polygons

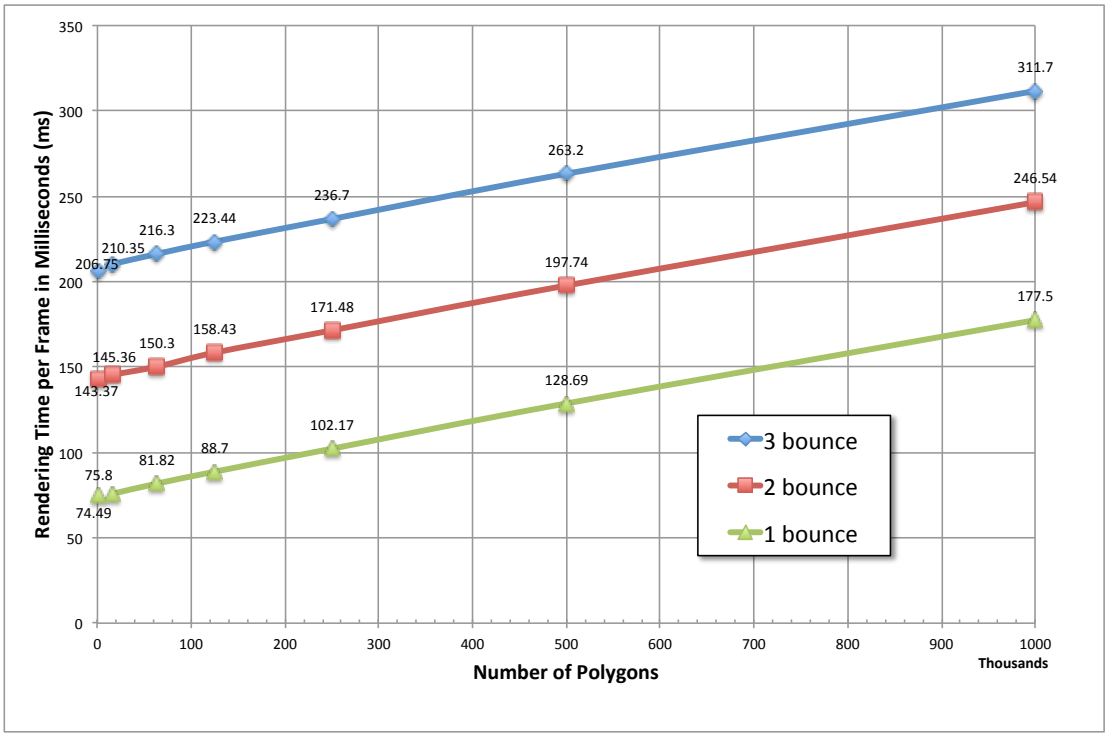


Figure 6.3: Rendering time by number of polygons (2K-1M triangles, 64 paths, 1-3 indirect bounces, 320 LHB directions, 128x128 LHB map size at 512x512 screen resolution).

Triangles	2K	4K	8K	16K	64K	125K	250K	500K	1000K
1 Bounce	74.49	74.55	75.00	75.80	81.82	88.70	102.17	128.69	177.50
2 Bounces	143.37	143.82	144.37	145.36	150.30	158.43	171.48	197.74	246.54
3 Bounces	206.75	208.43	209.33	210.35	216.30	223.44	236.70	263.20	311.70

Table 6.1: Individual timings shown in Figure 6.3 in milliseconds (ms).

Figure 6.3 shows the impact of the number of polygons on the rendering time. The plots show three path tracing renderings, with one, two and three bounces of indirect illumination computation on top of direct illumination computation. Direct illumination (which is L^1 in Equation 3.27) is computed using a eye-based ray and a shadow ray. One bounce of indirect illumination requires the generation of a randomly chosen reflected direction and computes the L^2 term in Equation 3.27. Each plot shows

that the relationship between the rendering time and the number of polygons is almost exactly linear. The proposed Monte-Carlo path tracer takes the same amount of time to compute more bounces in the rendering equation.

6.1.3 Number of Directions

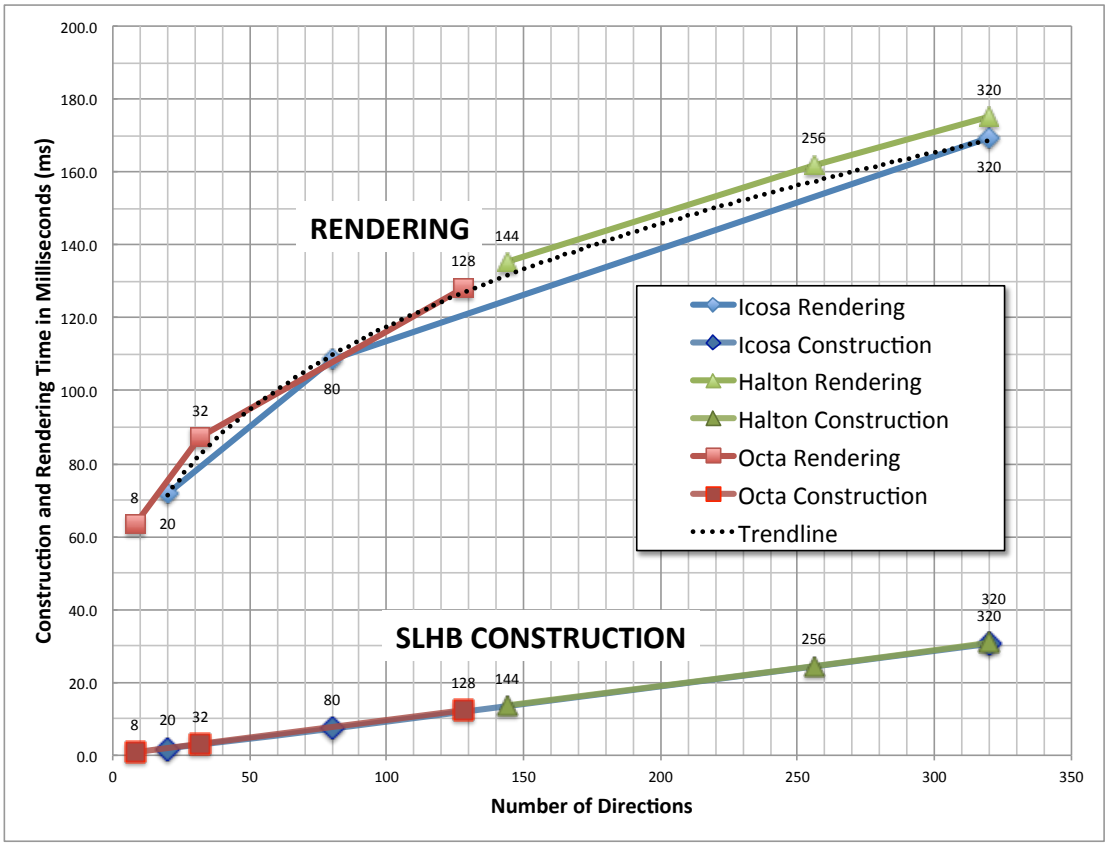


Figure 6.4: Rendering and construction time by increasing number of directions (250K Buddha model, 64 paths, 2 indirect bounces, 8-320 LHB directions, 128x128 LHB map size at 512x512 screen resolution).

Directions	Icosahedron 20	Icosahedron 80	Icosahedron 320
Time (ms)	71.63 (1.92)	108.67 (7.30)	169.48 (30.56)
Directions	Octahedron 8	Octahedron 32	Octahedron 128
Time (ms)	63.43 (0.83)	87.39 (3.15)	128.32 (12.36)
Directions	Halton 144	Halton 256	Halton 320
Time (ms)	135.38 (13.65)	162.07 (24.44)	175.27 (30.86)

Table 6.2: Individual timings shown in Figure 6.4 in milliseconds (ms).

In order to determine the scalability of the algorithm in terms of required computation time for a various number of directions, the 250K Buddha model was rendered under the same conditions as in the previous section, with three sets of directions. The average timings are given in Table 6.2. Three methods

have been selected to generate random directions over the sphere, which are Icosahedron subdivision, Octahedron subdivision and Halton sequence as described in Chapter 3. We have chosen 3 reasonable directions for each group. For the Icosahedron method, three levels of subdivision are chosen to generate 20, 80 and 320 directions. For the Octahedron method, 8, 32 and 128 directions are used. For the Halton sequence numbers, 144, 250 and 320 random samples are generated in 2D and projected onto spherical coordinates to create the random directions. The time measurement includes rasterizing scenes into LHB buffers (Construction time) with sizes of 128 x 128 pixels, and traces 64 rays per pixel for two bounces. The construction time is given in round brackets in Table 6.2. The construction time and overall rendering time of the three methods are plotted in Figure 6.4. The rendering time chart shows that each group has some linear characteristics. The dotted trend line for all methods represents the fact that the overall relationship between rendering time and the number of directions is linear. The non-linear rendering time occurs when the number of directions becomes small, as shown in the chart. A small number of directions may introduce uneven distributions on the sphere such that path-tracing takes longer to trace ray through the SLHB structure. However, the construction times of the three different methods have an exact linear time component to build the visibility structure as the number of direction increases. From the construction timing result, the average construction time to build the SLHB is almost 100 directions per 10ms in this example. In other words, the proposed software rasterization method is capable of rasterizing up to 2,500 million triangles per second (250K triangles * 100 directions / 10 ms) or computing 2,621 MRays/second (512x512 screen resolution * 100 directions / 10 ms). The higher the number of directions then the greater the accuracy of visibility that is obtained; however an increased number of directions requires more memory and longer times to build the SLHB.

6.1.4 Hit Buffer Size

Buffer Size	16x16	32x32	64x64	128x128	256x256
16K Triangles	132.70	136.29	139.07	143.74	155.05
64K Triangles	136.55	141.16	144.43	149.63	158.99
125K Triangles	142.68	147.27	150.95	156.95	166.38
250K Triangles	152.99	159.36	164.32	169.48	182.21
500K Triangles	175.72	182.99	190.02	196.19	210.11
1000K Triangles	220.11	228.55	237.47	244.80	257.74

Table 6.3: Individual timings shown in Figure 6.5 in milliseconds (ms).

The size of the Layered Hit Buffers is another important factor in determining the overall time for building the acceleration structure (SLHB). The smaller the Hit Buffers, the less time is required to construct the SLHB. However, decreasing the size will result in very coarse visibility structure. Figure 6.5 shows plots the construction and rendering of six Buddha models consisting of a various number of polygons, ranging from 16K to 1M. The higher the number of polygons, then the longer it takes to render the models, varying in a linear fashion. Although the number of triangles is different for each model, the

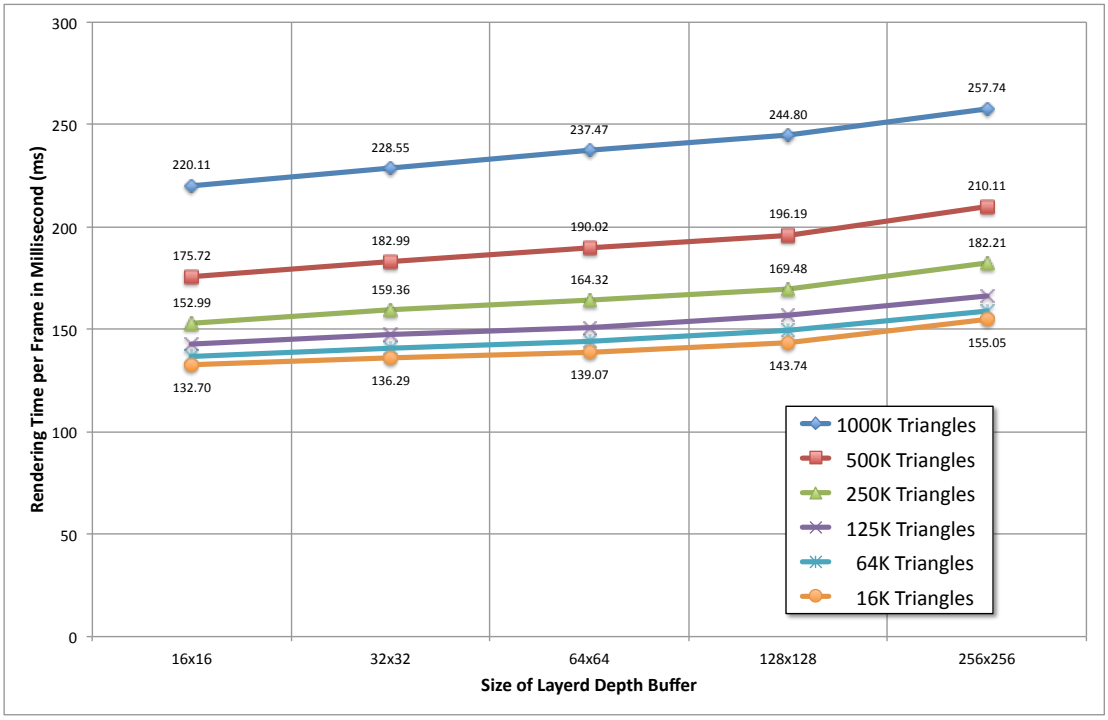


Figure 6.5: Rendering time by various size of Layered Hit Buffer (16K-1M triangles, 64 paths, 2 indirect bounces, 320 LHB directions, 16x16-256x256 LHB map size at 512x512 screen resolution).

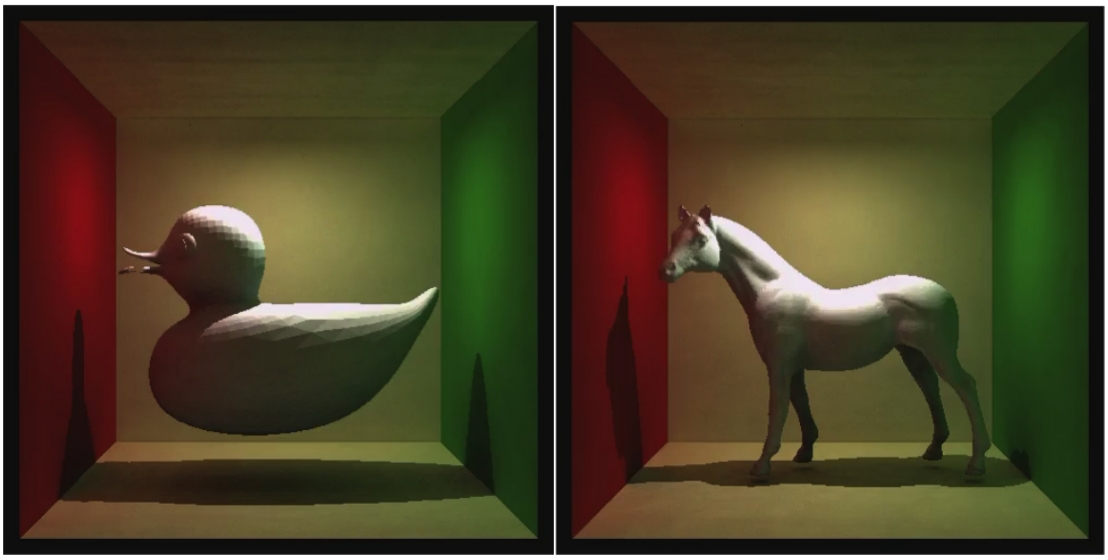
amount of area covered in the SLHB is similar to the test scene, since the models are in multi-resolution. Therefore, individual plots show the exact same slope, but located in a linearly higher position. Each plot has a very low slope, which means the rasterization has a small overhead for a larger size of LHB. The timing can be seen to be proportional to the covering area of the LHB, and the larger the size of the model, then the longer it takes to build. For following experiments, we were able to achieve plausible path-tracing results in real-time with only 128 to 320 LHB directions in a resolution of 128 x 128 for the Hit Buffer with 16 layers which is sufficient to capture all elements.

6.2 Performance Analysis of Complex Objects

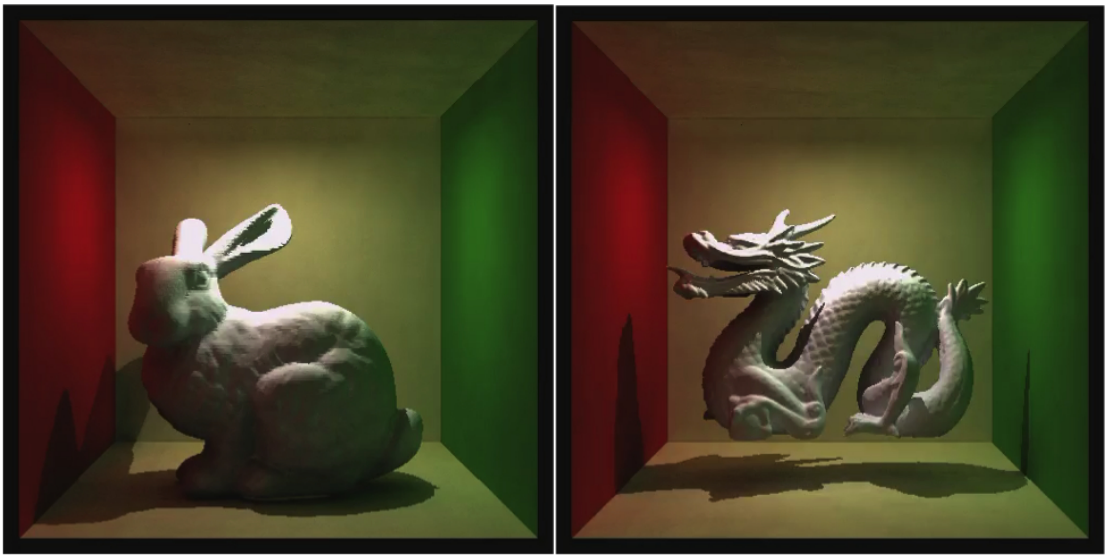
In this section, the scalability test is extended to measure the performance of the rendering method using SLHB, by applying the same number of 320 directions for all of the test scenes. This means the polygons are rasterized into 320 LHBs. For typical objects, it takes between 10-100 milliseconds to build the SLHB, and the memory consumption is dependent on the number of directions and the size of the hit buffer.

6.2.1 Rendering Timing of Various Objects

The depth complexity of the scenes affects the rendering timing in locating the nearest intersection. In order to check the rendering performance for various objects under a similar depth complexity, we have placed objects in a cornell-style room. Figure 6.6 shows rendered images of the duck, horse, bunny and dragon models illustrating the correct effects of global illumination. The scenes shows an example of a point light source from the ceiling towards the object in the middle such that direct lighting only



(a) Duck (14K) (b) Horse (100K)



(c) Bunny (70K) (d) Dragon (250K)

Figure 6.6: Rendered images of complex objects (resolution 512x512).

covers some part of the scene and generates sharp shadows, Other areas are illuminated from indirect inter-reflected lighting and show soft indirect shadow effects.

The statue model in Figure 6.7 is rendered at a high resolution with many paths. The rendering time takes 15,904 ms at a resolution of 1024 x 1024, with 640 paths per pixel for 5 bounces (see Table 6.4). This is approximately 42 million path rays per second, accounting for 5 bounces. This illustrates that the rendering solution presented here can cope with a highly complex model, and produce a high-quality image. The image shows that very limited direct lighting is available in the scene. The statue model is mainly lit by indirect illuminated energy, which shows the inter-reflections from other walls. The detail of performance measurements on various conditions is given in the following sections.

Figure 6.7: One million polygon statue rendered at 15,904ms per frame, which is 42M paths per frame (640 paths, 5 indirect bounces, 320 LHB directions, 128x128 LHB map size at 1024x1024 screen resolution).

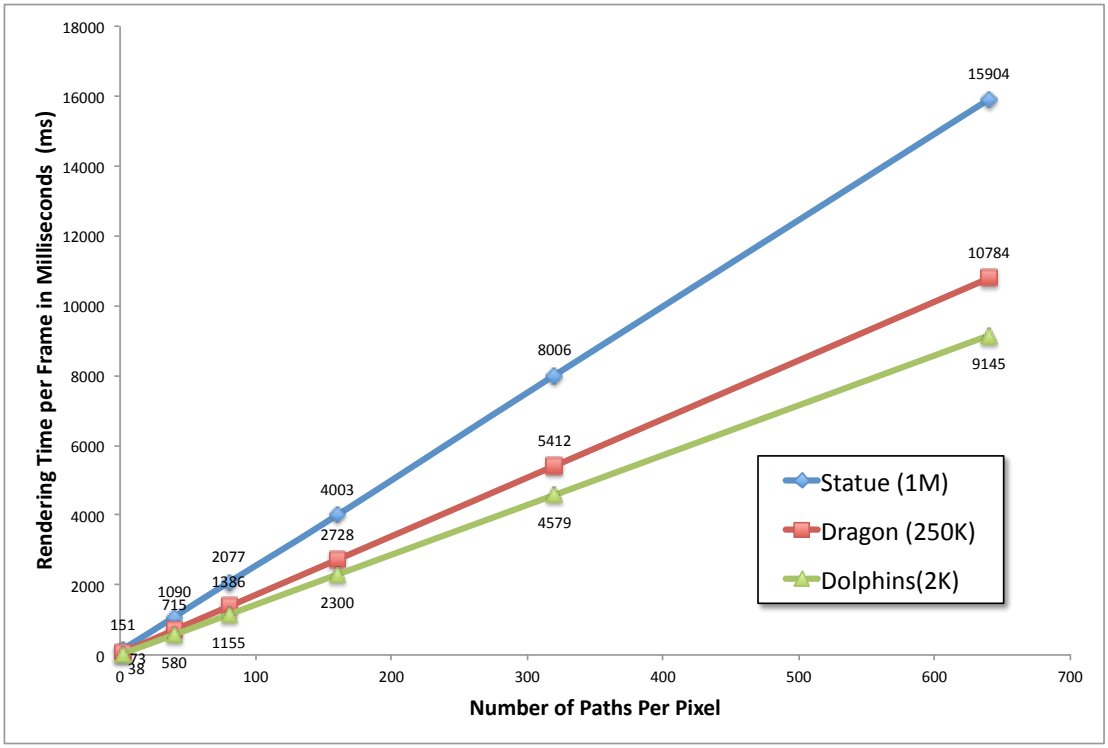


Figure 6.8: Paths VS frame rates by increasing number of paths (2K-1M triangles, 2-640 paths, 5 indirect bounces, 320 LHB directions, 128x128 LHB map size at 1024x1024 screen resolution).

Paths	2 Paths	10 Paths	40 Paths	80 Paths	160 Paths	320 Paths	640 Paths
Dolphins(2K)	38	152	580	1,155	2,300	4,579	9,145
Dragon (250K)	73	209	715	1,386	2,728	5,412	10,784
Statue (1M)	151	349	1,090	2,077	4,003	8,006	15,904

Table 6.4: Individual timings shown in Figure 6.8 in milliseconds (ms).

6.2.2 Paths VS Timing

Figure 6.8 shows plots of three scenes; the dolphin, dragon and statue models for a various number of paths per pixel. The rendering time is measured with a resolution of 1024 x 1024 with 5 indirect bounces. The rendering time grows linearly with the number of paths per pixel. Each plot shows a straight line with different slopes, where a higher slope represents a more complex object.

Figure 6.9 shows a statue scene, with timing ranging from 151 ms for 2 paths to 16 seconds for 640 paths. As shown in the figures, the new rendering method shows the same noise artifacts that appeared in the conventional path tracing results, when the number of paths per pixel is too low with a stratified sampling scheme. However, the noise becomes less noticeable as the number of paths increases.

6.3 Performance Comparison with OptiX path tracer

In this section, the performance of the new rendering solution is compared with other GPU-based works. The latest GPU-based ray tracing method [ZHWG08] reports that about 20 million rays can be traced in a

(a) 2 paths: 151ms (b) 5 paths: 226ms (c) 10 paths: 349ms

(d) 20 paths: 596ms (e) 40 paths: 1,090ms (f) 80 paths: 2,077ms

(g) 160 paths: 4,003ms (h) 320 paths: 8,006ms (i) 640 paths: 15,904ms

Figure 6.9: Timing measurements for various paths (1M statue model, 2-640 paths, 5 indirect bounces, 320 LHB directions, 128x128 LHB map size at 1024x1024 screen resolution).

dynamic scene, including a rebuild of the acceleration structure. That algorithm proves that an efficiently written GPU-based method could be 4 to 7 times faster than an optimized CPU algorithm [HMS06] and comparable with multiple CPU algorithms [MSK07]. However, the performance measurement does not account for random number generation. For this reason, a reference solution has been developed to measure accuracy timing using Optix SDK library [PBD+10]. Table 6.5 is a comparison of the result against [AL09, PBD+10]. However, those results are based on the measurement of intersection kernels only. Also, the measurement sees some benefit from arranging a large size of viewport to increase the coherence. As the figure shows, their measure is mainly based on primary ray and ambient rays, which

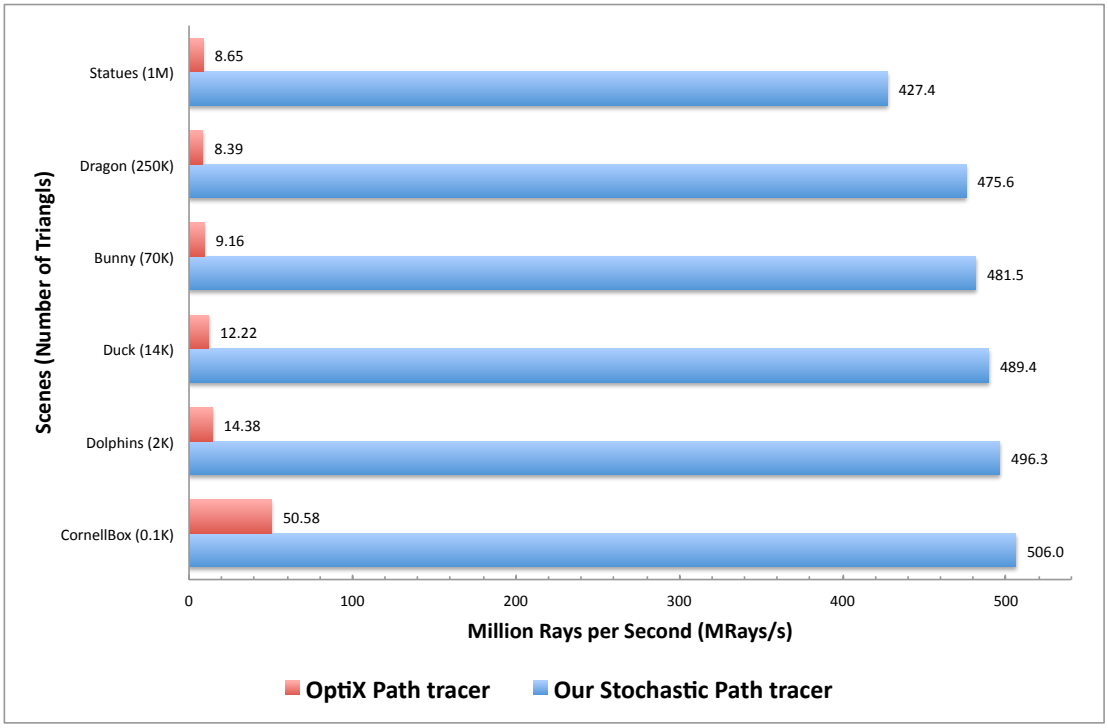


Figure 6.10: Performance comparison of OptiX [PBD+10] and the proposed CUDA Path tracer.

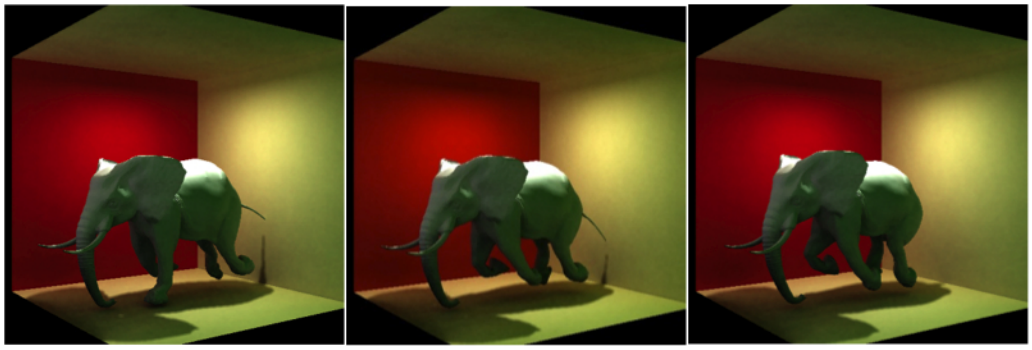
Model	Cornell Box	Dolphins	Duck	Bunny	Dragon	Statues
Triangles	100	2K	14K	70K	250K	1.0M
Stochastic(Precomputed)	506.0	496.3	489.4	481.5	475.6	427.4
OptiX Path tracer	50.58	14.38	12.22	9.16	8.39	8.65

Table 6.5: Individual performance shown in Figure 6.10 in millions of rays per second.

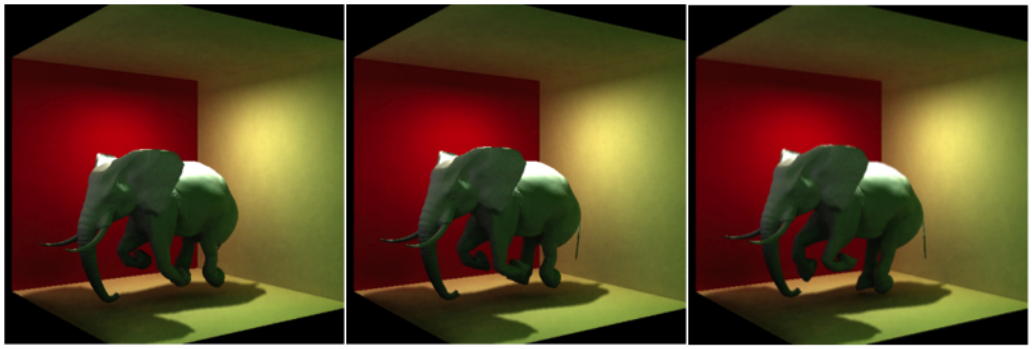
has a huge benefit over path tracing from random indirect rays. In this research, our measurement is based on random indirect rays where coherence is a minimum. In typical scenes, our new method is able to achieve about 450M rays per second whereas Optix-based ray tracing can send out only 8-50M rays per second. Figure 6.10 indicates that this new solution is a few orders of magnitude faster than the Optix-based path tracer [PBD+10] as the model becomes more complex. However, Optix-based ray tracing is capable of simulating general effects, whereas our renderer is optimized for environments where mostly diffuse objects are dominant.

6.4 Performance Analysis of Dynamic Elements

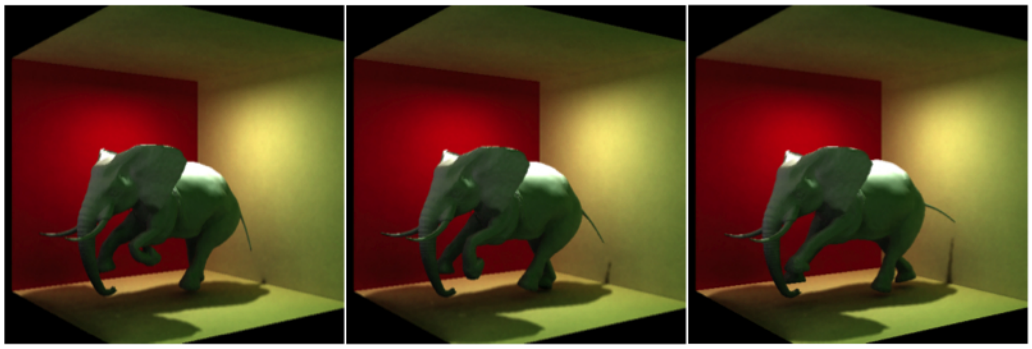
The performance details of the path tracing method for dynamic environments is described in this section. One of the main advantages of this new algorithm is the fact that the rebuilding cost of the visibility structure is very low compared with conventional acceleration structures for path tracing. This is because a parallel rasterization method has been predominantly used to build the structure on the fly. Figure 6.11 illustrates a sequence showing a galloping elephant model. In this example, the camera view is fixed



(a) frame 0 (b) frame 10 (c) frame 20



(d) frame 30 (e) frame 40 (f) frame 10



(g) frame 20 (h) frame 39 (i) frame 47

Figure 6.11: Dynamic movement of an elephant.

and the dynamic elephant object consisting of 49K polygons is updated in every frame. Figure 6.12 shows an enlarged image of the galloping elephant model, where only a small amount of direct light is illuminated on elephant's back. The rest of the elephant is indirectly illuminated from the walls, which shows green color bleeding. It also shows the presence of an umbra region, which is the dark part of the shadow where the direct light source is completely blocked by the elephant's occluding foot. The large penumbra region created by the elephant's body is also completely blocked from the direct lighting, but gathers indirect energy from all directions.

Figure 6.13 shows the SHLB construction time and rendering time of dynamic objects for a 48 camera frame sequence, and some rendered frames are shown in Figure 6.11. Three models have been chosen, which are of a horse, camel and elephant in the range of 17K to 85K polygons. The timing

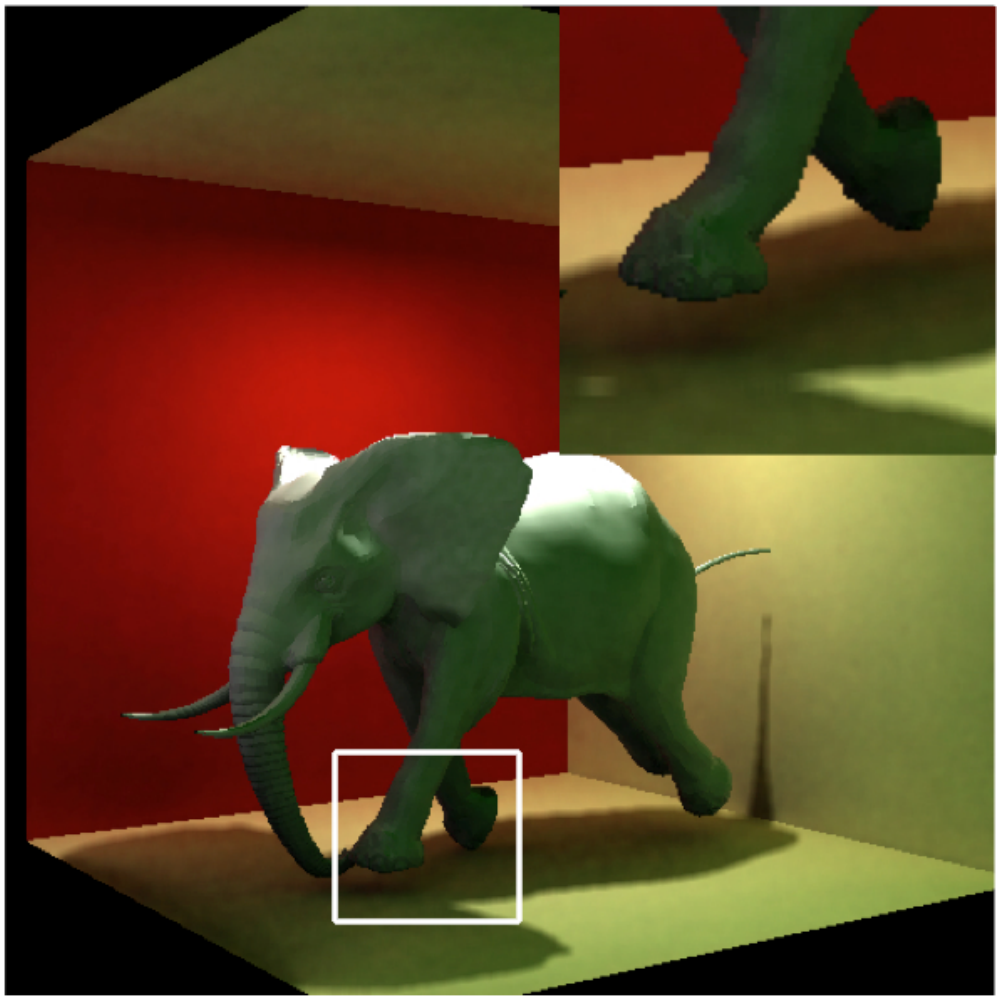


Figure 6.12: A large image of the galloping elephant model.

includes rebuilding an acceleration structure (construction time) and rendering one scene per frame. As shown in the figure, it takes 23 to 30 milliseconds to render an image at a resolution of 512 x 512 with 16 paths per pixel. Conventional path tracing algorithms suffer from the high cost of rebuilding the kd-tree, which prohibits real-time rendering of dynamic objects. However, the proposed solution takes 5 to 11 milliseconds to rebuild an acceleration structure per frame for tens of thousands of polygons. The SLHB construction and rendering time are consistent over all frames for each model. Although the dynamic elephant object (85K) has five times more polygons than the horse object (17K), there is only a slight additional rendering overhead (7ms). In other words, the cost of building a dynamic acceleration structure is relatively smaller than an overall rendering cost.

6.4.1 Performance comparison of Rebuilding the SLHB

The acceleration structure proposed in this thesis is capable of merging a few SLHBs efficiently. This function is very useful when a scene consists of a complex static environment with small dynamic objects. In this case, the SLHB is computed for the static scene once, which is reused for every frames. Only the SLHB for the dynamic objects is rebuilt, which is then appended to the static SLHB in order to complete the acceleration structure. In this way, only a small amount of time is required to update an

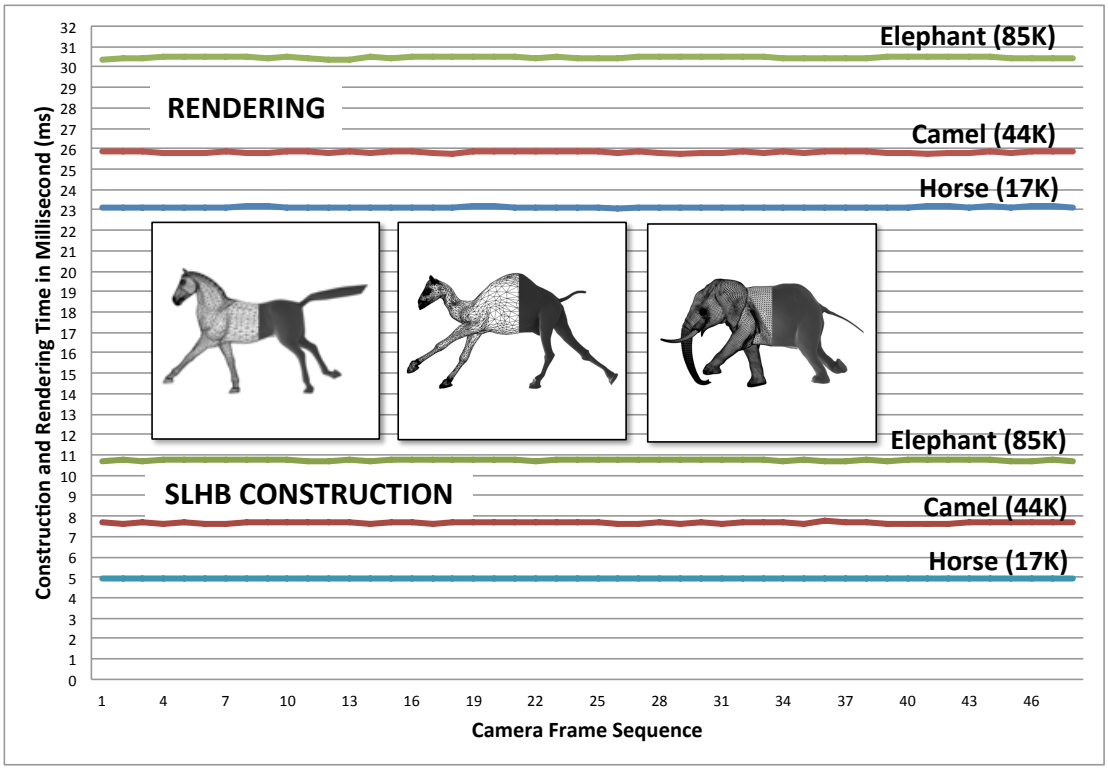


Figure 6.13: Rendering and construction time of dynamic objects at 512 x 512 resolution at 16 paths with 2 indirect bounces per pixel.

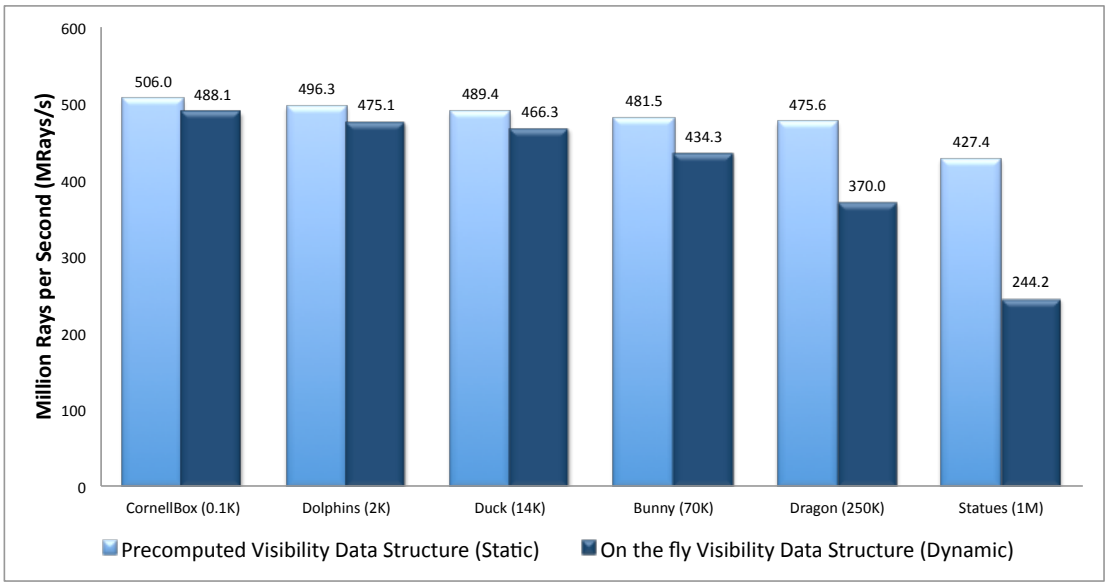


Figure 6.14: Performance comparison of static and dynamic data structure update.

acceleration structure efficiently. Figure 6.14 shows a performance comparison of the same scene in two conditions. One uses a pre-computed visibility structure, which is mainly used in a static environment in order to maximize the rendering performance. The other one rebuilds the acceleration structure for every frame, which is a technique typically used in a dynamic environment. The same test models are used

as in the previous section (Figure 6.6 and 6.7) to show how the performance degrades as the number of polygons increases for the two conditions. The bar charts show the rendering performance in millions of rays per second, for models containing 0.1K to 1M triangles. The pre-computed visibility method is capable of achieving a consistent performance over all the models, whereas the performance when using a dynamic acceleration structure degrades as the number of polygons increases, due to the cost of rebuilding the complex data structure. However, the performance of both methods is at least an order of magnitude faster than the current GPU-based path tracing method using a Optix SDK.

6.5 Scalability on Graphics Cards

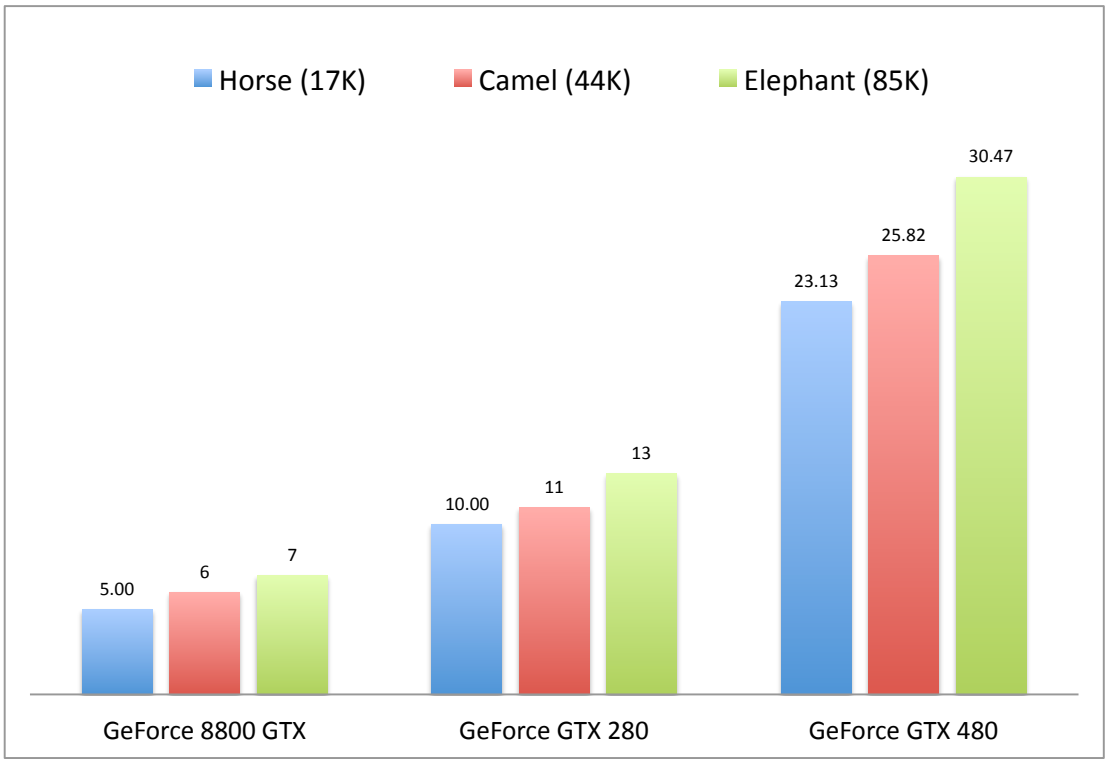


Figure 6.15: The scalability test on three graphics cards.

All the performance measurements in this thesis are based on the NVIDIA Geforce GTX 480, which has 480 stream processors. Figure 6.15 shows the rendering frame rates for three dynamic objects on various graphics cards. It indicates that the new rendering algorithm linearly scales on other graphics cards such as the Geforce GTX 280, which has 240 core processors, or the 8800 GTX, with 128 core processors. Due to compatibility issues, the algorithm can not run on less than the CUDA 1.3 specification architecture.

6.6 Summary

In this chapter the evaluation of the proposed global illumination rendering method using rasterization on the GPU has been discussed. The results show that this new rendering method can render high-quality images efficiently with varying numbers of polygons and on different model sizes. This new algorithm

runs at real-time rates on modern graphics hardware, and it has also been shown that real-time rendering of dynamic objects is feasible.

Chapter 7

Conclusions and Future Works

In recent years, research into the field of global illumination has changed the focus of study towards real-time rendering systems. This is due to the considerable technical development of flexible modern graphics hardware, which has opened up a new research area in hardware-accelerated global illumination algorithms. Several methods have been presented to demonstrate GPU-based algorithms running entirely on graphics hardware, instead of using a traditional graphics pipeline. Although these methods are capable of running in real-time, they are mainly limited to static environments due to the high cost of rebuilding acceleration structures. This research has presented an efficient visibility acceleration structure using a customized CUDA rasterization, so that the proposed path tracing method is capable of rendering global illumination for dynamic environments at real-time frame rates. The main topics investigated in this thesis are summarized in Section 7.1, and there is a discussion on the recommended direction for ongoing and future research in Section 7.2 .

7.1 Summary

The main objective of this research was to develop a novel method for enabling real-time rendering of physically simulated global illumination in dynamic virtual environments, accounting for changes of view, material, lighting and objects. In order to achieve this goal, the concept of adapting an approximated visibility technique for global illumination was first introduced. Our perceptual study [YCK+09] indicates that some visibility approximations can yield rendering results that are considered to be as realistic as accurate solutions. Based on this fact, a new parameterization of an acceleration structure was proposed, which holds approximated visibility information in multi-layered hit buffers. The hit buffers are a similar notion to those introduced in Layered Depth Images [SGHS98]. However, the data structure presented in this study extends LDI concept to spherical directions in order to build a 5D visibility field, so that any intersections in a scene are approximated in a simple depth search manner. In order to construct the Spherical Layered Hit Buffers (SLHB), a new and efficient CUDA-based rasterization method was proposed, due to the fact that OpenGL rasterization is not capable of generating structural buffers. This research adapted our proved concept [YCK+09] that direct illumination requires higher accuracy than indirect illumination computation. Therefore, direct illumination is computed directly from the light source using a large shadow map, whereas indirect illumination is computed by the SLHB to

determine the approximated visibility information. It has been shown that the performance of this new rasterization is comparable to the standard OpenGL rasterizer, and that in some cases this new rasterization method is even faster. The proposed CUDA rasterizer has been used to build the SLHB on the fly so that the visibility field is updated every frame. In the rendering stage, a final viewing is rendered in real-time from any point in a virtual environment. Two methods are proposed to evaluate the rendering equation. One uses a deterministic gathering method to compute the Monte Carlo Integration of the light transport in a parallel fashion. The other method uses stochastic methods, which employs an important sampling scheme to approximate fast indirect illumination using approximated visibility. The fundamental achievement of this research is the development of a CUDA rasterizer, parameterization of an approximated visibility field and a real-time GPU Monte-Carlo path tracing solution.

7.2 Future Research

There are several possible areas for future research. Within these research directions, some ideas will be described regarding how to use the new rasterization and SLHB structure in the field of computer graphics. In the following section, the ongoing works of this research will also be presented, focusing on real-time global illumination in virtual reality in Section 7.2.1 and augmented reality in Section 7.2.2.

Figure 7.1: An example of McGuire's [MESL10] work, showing the difference between conventional and stochastic rasterization.

The new CUDA rasterizer is an optimized software rasterization method on a GPU, which enables manipulation of a rasterized output in a structural data format. In this algorithm, the CUDA rasterizer has been used to build multiple layered hit buffers, with depth and polygonal information that serve as a discrete acceleration structure for global illumination. Although this user-defined graphics pipeline is very useful, there are several possible improvements over the proposed CUDA rasterization method in order to achieve complex tasks such as stochastic rasterization, motion blur and depth of field effects. McGuire et al [MESL10] presented a stochastic rasterization on GPU architectures, achieving interactive performance for complex scenes as shown in Figure 7.1.

A brute force way of achieving this effect is to render multiple conventional images using a standard rasterization function as a pinhole camera, and then applying post-processing to average the pixel values [HA90, WGER05]. The accumulation buffering is an inefficient method and could result in a discrete ghosting effect. The advantage of stochastic rasterization methods [AMMH07, FLB+09, HQL+10] is the ability to use a conventional rasterizer to check many visibility tests over time and for different lens positions, while computing one shading value per pixel. In a similar way, the proposed CUDA rasterizer

can be extended to perform multi-view rasterization by loading polygon data once and rasterizing over many different perspective views in a single graphics pipeline. This means it avoids multiple loading of scene data and transfer between scenes in the GPU for large models. In this way, the accumulation buffering can be simulated for multiple views, while maintaining a shading process at a minimum per pixel.

Figure 7.2: An example of lens blur effect from Lee's work [LES09].

Another technique that benefits from multiple view rasterization is the lens blur effect. Lee et al [LES09, LES10] presented a GPU-based solution to simulate depth of field effects by using a layered image-based scene. Most real-time approaches for the lens effect use a single view image to avoid multiple rendering of a scene, and simulate approximated visibility. However, Lee's method uses a multiple render target function in the graphics hardware to generate multiple layered images in order to represent a scene. These layered images are placed in several locations towards a camera view at an exponential distance. Then the multiple layered images are merged into a single image by applying cone filtering to multiple layers of the depth buffer, thereby yielding the depth of field effect. However, their method can only rasterize a certain number of layers at once due to the limitations of the OpenGL hardware, whereas the CUDA rasterization method defined in this research study has no limit in creating a number of layers, as long as memory is available. This new solution can be extended to support lens blur in a similar way. An LHB structure has depth and polygon ids which are very similar to Lee's layered image representation. One advantage is that the SLHB holds layered depth buffer in many directional views, so that the SLHB can be built once and used for any camera view to show the depth of field effects.

There are some other applications that can benefit from the new acceleration structure defined in this research study, for example, instant radiosity algorithm [Kel97] that places many virtual point lights in a scene to approximate global illumination. However, generating shadow maps for each VPL is too

costly. Instead of generating many shadow maps, the SLHB acceleration structure provides approximated visibility for faster occlusion query.

7.2.1 Real-time Global illumination in Virtual Reality

Global illumination in the context of Virtual Reality (VR) is another area that can benefit from a real-time rendering system. Global illumination of virtual environments could enhance the visual realism and the sense of presence. However, due to the computational complexity of rendering systems, many VR applications are limited to a direct illumination model or pre-computed global illumination for static scenes. Dmitriev et al [DAK+04] have applied pre-computed radiance transfer [SKS02] to VR rendering in order to support global illumination in a CAVE (Cave Automatic Virtual Environment) system. In our earlier research, we also also adapted the Virtual Light Field (VLF) [SMKY04] technique to provide global illumination for VR applications [MKYS07a, MYK+08]. However, the rendering method employed a hybrid technique to provide global illumination, by integrating pre-computed global illumination with simple dynamic elements on the fly. The global illumination effects for static elements of the scene are from pre-computed VLF data, whereas dynamic elements, such as avatar movement, soft shadows and reflection of avatar are added to a rendered image in post-processing for each frame. This is because the rendering cost of the full global illumination prohibits real-time rates. The new GPU-based path tracing solution devised in our research can be employed to overcome the problems regarding dynamic elements, allowing realistic virtual environments.

Figure 7.3: Real-time Global illumination in Virtual Reality.

Figure 7.3 shows a photograph of a participant in a CAVE virtual library. The scene is rendered with pre-computed VLF, which presents the dynamic virtual avatar in the mirror by tracking data from

two motion sensors. We also conducted some perceptual studies [SKMY09] [YMKS11] in order to understand whether physically-based global illumination within dynamic environments could enhance the sense of presence in an immersive virtual reality environment. The results indicate that visual realism enhances a realistic response in an immersive virtual environment.

7.2.2 Real-time Global Illumination in Augmented Reality

(a) HDR Environment Lighting

(b) Real Scene

(c) Virtual Statue in a Real Scene

Figure 7.4: Real-time Global Illumination in Augmented Reality.

Augmented Reality (AR) techniques offer a method to merge virtual environments into real world environments. AR is also related to the term 'mixed reality', where virtual objects are seamlessly presented in the physical world, which are often indistinguishable from realistic photos. Many researchers have studied the way that shadows influence [SKT03] the presence of virtual objects in an augmented scene. In recent years, some mixed reality works [Gro05, GEM07] have studied the influence of inter-

reflection between virtual and real objects. Most AR applications simply merge virtual objects in static photo-realistic images, which often lack dynamic environmental illumination and interactive indirect illumination. The new real-time path tracer in our research could provide a framework for mixed reality, allowing a color bleeding effect at real-time rates.

Figure 7.4 shows a typical example of an augmented reality scene, where two realistic objects on the table are illuminated from many different light sources, and a virtual statue is rendered using captured environmental lighting. The small image at the top right corner in Figure 7.4 shows shadow and shading of the virtual object before it is integrated into the real environment.

In this ongoing AR study, it is the aim to provide seamless compositing of dynamic virtual objects into real scenes, considering inter-reflection from surrounding real objects. In order to achieve this, a perceptual study is conducted to measure realism, in order to speed up the rendering by allocating computational tasks where they are perceptually required the most. The initial focus is on investigating the perceptual metrics of measuring the realism of shadow, direct and indirect illumination. Then the perpetual metric is used to determine the appearance of virtual objects. In this way, the influence of individual components is measured so that more resources are allocated to where they are most important.

Appendix A

Summary of Notations

A.1 Geometry

SYMBOL	DESCRIPTION
x	Point(or position) in 3D space
S, A	A set of surface points
N_x	Normal at surface point x (Normalized $\|n\| = 1$)
dA_x	Differential surface area at point x
r_xy	Distance between surface points x and y
$\overline{xy}$	Direction from x to y
$\Theta, -\Theta$	Outgoing direction and its negate direction
$\Psi, -\Psi$	Incoming direction and its negate direction
ω	Direction (unit vector)
ω_i	Incoming or incident direction
ω_o	Outgoing or exitant direction
(θ, ϕ)	Direction in spherical coordinates
$d\omega$	Differential Solid Angle ($d\omega = \sin\theta d\theta d\phi$)
$d\omega_\Psi$	Differential Solid Angle in direction Ψ
Ω	Set of directions on the hemisphere
$\Omega_{4\pi}$	Set of directions on the unit sphere
$V(x,y)$	visibility function; $V(x,y) = \begin{cases} 1, & \text{if } x \text{ and } y \text{ are mutually visible;} \\ 0, & \text{otherwise.} \end{cases}$
$G(x,y)$	geometry term ($\frac{\cos(\theta_1)\cos(\theta_2)}{\|x-y\|^2}$)

A.2 Probability (Monte Carlo)

SYMBOL	DESCRIPTION
$\xi_1, \xi_2, ...\xi_n$	Canonical Uniformly Sampled Random Variables
$E[\xi]$	Expected value of random variable ξ
$\langle I \rangle$	Estimator of I
$\langle I \rangle_N$	N-sample estimate or I
$p(x)$	Probability density function(PDF)

A.3 Radiometry

SYMBOL	DESCRIPTION
Φ	Radiant flux (power) [(W)att]
E	Irradiance (flux are density) $[W \cdot m^{-2}]$
I	Radiant intensity (flux density per solid angle) $[W \cdot sr^{-1}]$
Q	Radiant Energy [(J)oule]
M, B	Radiant exitance, radiosity $[W \cdot m^{-2}]$
L	Radiance (flux density per area per solid angle)
L_e	Emitted Radiance
L_i	Incident Radiance
L_r	Reflected Radiance
$L(x \rightarrow \Theta)$	Radiance at x in outgoing direction Θ
$L(x \leftarrow \Psi)$	Radiance at x in incoming direction Ψ
ρ	Reflectance
ρ_d	Diffuse reflectance
ρ_s	Specular reflectance
$f_r(\Theta \leftrightarrow \Psi)$	BRDF(bidirectional reflectance distribution function)
$f_r(x, \Theta \leftrightarrow \Psi)$	BRDF at surface point x
σ_a	Absorption coefficient
σ_s	Scattering coefficient
σ_t	Attenuation coefficient

A.4 Miscellaneous

SYMBOL	DESCRIPTION
L	Light source
E	Eye or camera position
D	Diffuse reflection
G	Glossy reflection
S	Specular reflection
$\|$	Operator 'or'
$+$	Operator 'addition'
$\delta(x)$	Dirac delta distribution
ε	Error symbol

Appendix B

List of Publications

- **Visual Realism Enhances Realistic Response in An Immersive Virtual Environment - Part 2** I. Yu, J. Mortensen, P. Khanna, M. Slater *IEEE Computer Graphics and Applications, to appear*, 2011
- **Display-aware Image Editing** W.K. Jeong, M.K. Johnson, I. Yu, J. Kautz, H. Pfister, S. Paris *International Conference on Computational Photography*, 2011
- **Perceptual influence of approximate visibility in indirect illumination** I. Yu, A. Cox, M.H. Kim, T. Ritschel, T. Grosch, C. Dachsbacher, J. Kautz *ACM Transactions on Applied Perception*, 2009
- **Visual Realism Enhances Realistic Response in An Immersive Virtual Environment** M. Slater, P. Khanna, J. Mortensen, I. Yu *IEEE Computer Graphics and Applications*, 2009
- **Real-Time Global Illumination for VR Applications** J. Mortensen, I. Yu, P. Khanna, M. Slater, F. Tecchia, B. Spanlang, G. Marino, M. Slater *IEEE Computer Graphics and Applications*, 2008
- **Real-time Global Illumination in the CAVE** J. Mortensen, P. Khanna, I. Yu, M. Slater *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, 2007
- **A Non-parametric Guide for Radiance Sampling in Global Illumination** P. Khanna, M. Slater, J. Mortensen, I. Yu *CGIV '07: Proceedings of the Computer Graphics, Imaging and Visualisation*, 2007
- **A Visibility Field for Ray Tracing** J. Mortensen, P. Khanna, I. Yu, M. Slater *CGIV '07: Proceedings of the Computer Graphics, Imaging and Visualisation*, 2007
- **Presence in Response to Dynamic Visual Realism: A Preliminary Report of An Experiment Study** P. Khanna, I. Yu, J. Mortensen, M. Slater *VRST '06: Proceedings of the ACM symposium on Virtual reality software and technology*, 2006
- **A Virtual Light Field Approach to Global Illumination** M. Slater, J. Mortensen, P. Khanna, I. Yu, *CGI '04: Proceedings of the Computer Graphics International*, 2004

Bibliography

[AB91] E. H. Adelson and J. R. Bergen. The plenoptic function and the elements of early vision. MIT Press, 1991.

[AFO05] Okan Arikan, David A. Forsyth, and James F. O'Brien. Fast and detailed approximate global illumination by irradiance decomposition. *ACM Trans. Graph.*, 24:1108–1114, July 2005.

[AK87] James Arvo and David Kirk. Fast ray tracing by ray classification. *Proc.of SIGGRAPH 1987*, 21(4):55–64, July 1987.

[AK90] James Arvo and David Kirk. Particle transport and image synthesis. *Computer Graphics*, 24(4):53–66, 1990.

[AL09] Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on gpus. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 145–149, New York, NY, USA, 2009. ACM.

[AMMH07] Tomas Akenine-Möller, Jacob Munkberg, and Jon Hasselgren. Stochastic rasterization using time-continuous triangles. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS symposium on Graphics hardware*, pages 7–16, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[AW87] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87*, pages 3–10. Elsevier Science Publishers, Amsterdam, North-Holland, 1987.

[Bad90] Didier Badouel. An efficient ray-polygon intersection. In *Graphics Gems*, pages 390–393. Academic Press, 1990.

[BCL+07] Louis Bavoil, Steven P. Callahan, Aaron Lefohn, João L. D. Comba, and Cláudio T. Silva. Multi-fragment effects on the gpu using the k-buffer. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 97–104, New York, NY, USA, 2007. ACM.

[Bit99] Jiri Bittner. Hierarchical techniques for visibility determination. Technical Report DS-005, Department of Computer Science and Engineering,

Czech Technical University in Prague, March 1999. Also available as http://www.cgg.cvut.cz/~bittner/publications/minimum.ps.gz.

[BN76] James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Commun. ACM*, 19(10):542–547, 1976.

[BUN05] M. BUNNELL. *GPU Gems 2*, chapter Dynamic Ambient Occlusion and Indirect Lighting, pages 615–633. Addison Wesley, 2005.

[Car84] Loren Carpenter. The a -buffer, an antialiased hidden surface method. *SIGGRAPH Comput. Graph.*, 18:103–108, January 1984.

[CB04] Per H. Christensen and Dana Batali. An irradiance atlas for global illumination in complex production scenes. In *Rendering Techniques*, pages 133–142, 2004.

[CBCG02] Wei-Chao Chen, Jean-Yves Bouguet, Michael H. Chu, and Radek Grzeszczuk. Light field mapping: efficient representation and hardware rendering of surface light fields. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 447–456, New York, NY, USA, 2002. ACM Press.

[CCWG88] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. *SIGGRAPH Comput. Graph.*, 22(4):75–84, 1988.

[CDP95] Frédéric Cazals, George Drettakis, and Claude Puech. Filtering, clustering and hierarchy construction: a new solution for ray-tracing complex scenes. *Computer Graphics Forum*, 14(3):371–382, 1995.

[CF99] Emilio Camahort and Don Fussell. A geometric study of light field representations. Technical report tr99-35, Department of Computer Sciences, The University of Texas at Austin, 1999.

[CG85] Michael F. Cohen and Donald P. Greenberg. The hemi-cube: a radiosity solution for complex environments. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 31–40, New York, NY, USA, 1985. ACM Press.

[CHCH06] Nathan A. Carr, Jared Hoberock, Keenan Crane, and John C. Hart. Fast gpu ray tracing of dynamic meshes using geometry images. In *GI '06: Proceedings of Graphics Interface 2006*, pages 203–209, Toronto, Ont., Canada, Canada, 2006. Canadian Information Processing Society.

[Che90] Shenchang Eric Chen. Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system. volume 24, pages 135–144, August 1990.

[Che95] Shenchang Eric Chen. Quicktime vr — an image-based approach to virtual environment navigation. *Computer Graphics*, 29(Annual Conference Series):29–38, 1995.

[CHH02] Nathan A. Carr, Jesse D. Hall, and John C. Hart. The ray engine. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 37–46, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[CHH03] Nathan A. Carr, Jesse D. Hall, and John C. Hart. Gpu algorithms for radiosity and subsurface scattering. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 51–59, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[CHL04] Greg Coombe, Mark J. Harris, and Anselmo Lastra. Radiosity on graphics hardware. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 161–168, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[Chr99] Per H. Christensen. Faster photon map global illumination. *J. Graph. Tools*, 4(3):1–10, 1999.

[Chr05] Martin Christen. *Ray Tracing on GPU*. Diploma thesis, University of Applied Sciences Basel, Switzerland, 2005.

[CLF98] Emilio Camahort, Apostolos Lerios, and Donald Fussell. Uniformly sampled light fields. In G. Drettakis and N. Max, editors, *Rendering Techniques '98 (Proceedings of Eurographics Rendering Workshop '98)*, pages 117–130, New York, NY, 1998. Springer Wien.

[Coh94] Daniel Cohen. Voxel traversal along a 3d line. In Paul Heckbert, editor, *Graphics Gems IV*, page 366 369. Academic Press, 1994.

[Coo86] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, January 1986.

[CPC84] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, volume 18, pages 137–145, New York, NY, USA, July 1984. ACM Press.

[CPWAP08] Ewen Cheslack-Postava, Rui Wang, Oskar Akerlund, and Fabio Pellacini. Fast, realistic lighting and material design using nonlinear cut approximation. *ACM Trans. Graph.*, 27:128:1–128:10, December 2008.

[CRMT91] Shenchang Eric Chen, Holly E. Rushmeier, Gavin Miller, and Douglass Turner. A progressive multi-pass method for global illumination. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 165–174, New York, NY, USA, 1991. ACM Press.

[CSS96] Per H. Christensen, Eric J. Stollnitz, and David H. Salesin. Global illumination of glossy environments using wavelets and importance. *ACM Transactions on Graphics*, 15(1):37–71, 1996.

[Cud11] *NVIDIA CUDA C Programming Guide*. NIVIDIA CUDA, 2011.

[CW93a] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. *Computer Graphics*, 27(Annual Conference Series):279–288, 1993.

[CW93b] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Boston, MA, 1993.

[DAK+04] Kirill Dmitriev, Thomas Annen, Grzegorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. A cave system for interactive modeling of global illumination in car interior. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '04, pages 137–145, New York, NY, USA, 2004. ACM.

[DDSC11] Kurt Debattista, Piotr Dubla, Luis Santos, and Alan Chalmers. Wait-free shared-memory irradiance caching. *IEEE Comput. Graph. Appl.*, 31:66–78, September 2011.

[DS05] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 203–231, New York, NY, USA, 2005. ACM.

[DS06] Carsten Dachsbacher and Marc Stamminger. Splatting indirect illumination. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 93–100, New York, NY, USA, 2006. ACM.

[DTM96] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Computer Graphics*, 30(Annual Conference Series):11–20, 1996.

[Dut96] Philip Dutré. *Mathematical Frameworks and Monte Carlo Algorithms for Global Illumination in Computer Graphics*. PhD thesis, Katholieke Universiteit Leuven, September 1996.

[FBG02] Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Local illumination environments for direct lighting acceleration. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 7–14, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[FLB+09] Kayvon Fatahalian, Edward Luong, Solomon Boulos, Kurt Akeley, William R. Mark, and Pat Hanrahan. Data-parallel rasterization of micropolygons with defocus and motion blur. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 59–68, New York, NY, USA, 2009. ACM.

[FS05] Tim Foley and Jeremy Sugerman. Kd-tree acceleration structures for a gpu raytracer. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 15–22, New York, NY, USA, 2005. ACM Press.

[GEM07] Thorsten Grosch, Tobias Eble, and Stefan Mueller. Consistent interactive augmentation of live camera images with correct near-field illumination. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, VRST '07, pages 125–132, New York, NY, USA, 2007. ACM.

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1996. ACM Press.

[GKBP05] Pascal Gautron, Jaroslav Krivanek, Kadi Bouatouch, and Sumanta N. Pattanaik. Radiance cache splatting: A gpu-friendly global illumination algorithm. In *Rendering Techniques 2005, Eurographics Symposium on Rendering*, pages 55–64, June 2005.

[GKPB04] Pascal Gautron, Jaroslav Krivanek, Sumanta Pattanaik, and Kadi Bouatouch. A novel hemispherical basis for accurate and efficient rendering. In *Rendering Techniques 2004: 15th Eurographics Workshop on Rendering*, pages 321–330, June 2004.

[Gla84] Andrew S. Glassner. Space subdivision for fast ray tracing. volume 4, pages 15–22, 1984.

[Gla89] Andrew S. Glassner. *An Introduction to Ray tracing*. Academic Press, San Diego, CA, USA, January 1989.

[Gre86] Ned Greene. Environment mapping and other applications of world projections. *IEEE Comput. Graph. Appl.*, 6(11):21–29, 1986.

[Gro05] Thorsten Grosch. Differential photon mapping - consistent augmentation of photographs with correction of all light paths. 2005.

[GSCH93] Steven J. Gortler, Peter Schröder, Michael F. Cohen, and Pat Hanrahan. Wavelet radiosity. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 221–230, New York, NY, USA, 1993. ACM Press.

[GSHG98] Gene Greger, Peter Shirley, Philip M. Hubbard, and Donald P. Greenberg. The irradiance volume. *IEEE Computer Graphics & Applications*, 18(2):32–43, March 1998.

[GTGB84] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 213–222, New York, NY, USA, 1984. ACM Press.

[HA90] Paul Haeberli and Kurt Akeley. The accumulation buffer: hardware support for high-quality rendering. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '90, pages 309–318, New York, NY, USA, 1990. ACM.

[Hav01] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Czech Technical University in Prague, Prague, April 2001.

[HG86] Eric A. Haines and Donald P. Greenberg. The light buffer: A ray tracer shadow testing accelerator. *IEEE Computer Graphics and Applications*, 6(9):6–16, September 1986.

[HLCS99] W. Heidrich, H. Lensch, M. Cohen, and H. Seidel. Light field techniques for reflections and refractions. 1999.

[HMS06] Warren Hunt, William R. Mark, and Gordon Stoll. Fast kd-tree construction with an adaptive error-bounded heuristic. In *2006 IEEE Symposium on Interactive Ray Tracing*. IEEE, Sept. 2006.

[HQL+10] Qiming Hou, Hao Qin, Wenyao Li, Baining Guo, and Kun Zhou. Micropolygon ray tracing with defocus and motion blur. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 64:1–64:10, New York, NY, USA, 2010. ACM.

[HSA91] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 197–206, New York, NY, USA, 1991. ACM Press.

[IMG00] Aaron Isaksen, Leonard McMillan, and Steven J. Gortler. Dynamically reparameterized light fields. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 297–306, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[IPL97] Insung Ihm, Sanghoon Park, and Rae Kyoung Lee. Rendering of spherical light fields. In *PG '97: Proceedings of the 5th Pacific Conference on Computer Graphics and Applications*, page 59, Washington, DC, USA, 1997. IEEE Computer Society.

[JC98] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. pages 311–320, July 1998.

[Jen95] Henrik Wann Jensen. Importance driven path tracing using the photon map. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 326–335, New York, NY, 1995. Springer-Verlag.

[Jen96] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 21–30, London, UK, 1996. Springer-Verlag.

[Jen97] Henrik Wann Jensen. Rendering caustics on non-Lambertian surfaces. *Computer Graphics Forum*, 16(1):57–64, 1997.

[JMLH01] Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. A practical model for subsurface light transport. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 511–518, New York, NY, USA, August 2001. ACM Press.

[JW89] D. Jevans and B. Wyvill. Adaptive voxel subdivision for ray tracing. In *Proc. Graphics Interface*, pages 164–172, 1989.

[Kaj86] James T. Kajiya. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA, 1986. ACM Press.

[KBR04] John Kessenich, Dave Baldwin, and Randi Rost. *The OpenGL Shading Language 1.1*. 3Dlabs, Inc. Ltd., April 2004. Document Revision 59.

[Kel97] Alexander Keller. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[Kel98] Alex Keller. *Quasi-Monte Carlo Methods for Photorealistic Image Synthesis*. Diploma thesis, University Kaiserslautern, 1998.

[KGBP05] Jaroslav Křivánek, Pascal Gautron, Kadi Bouatouch, and Sumanta Pattanaik. Improved radiance gradient computation. In *SCCG '05: Proceedings of the 21st spring conference on Computer graphics*, pages 155–159, New York, NY, USA, 2005. ACM Press.

[KK86] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 269–278, New York, NY, USA, 1986. ACM Press.

[KL04] Filip Karlsson and Carl Johan Ljungstedt. *Ray Tracing Fully Implemented on Programmable Graphics Hardware*. M.s. thesis, Chalmers University of Technology Gteborg, Sweden, 2004.

[KMYS04] Pankaj Khanna, Jesper Mortensen, Insu Yu, and Mel Slater. Fast ray tracing of scenes with unstructured motion. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Posters*, page 35, New York, NY, USA, 2004. ACM.

[KP98] Laszlo Szirmay Kalos and Werner Purgathofer. Global ray-bundle tracing with hardware acceleration. In *In Rendering Techniques '98*, pages 247–258, 1998.

[KS97] Krzysztof S. Klimaszewski and Thomas W. Sederberg. Faster ray tracing using adaptive grids. *IEEE Comput. Graph. Appl.*, 17(1):42–51, 1997.

[KSMY07] Pankaj Khanna, Mel Slater, Jesper Mortensen, and Insu Yu. A non-parametric guide for radiance sampling in global illumination. In *CGIV '07: Proceedings of the Computer Graphics, Imaging and Visualisation*, pages 41–48, Washington, DC, USA, 2007. IEEE Computer Society.

[KYMS06] Pankaj Khanna, Insu Yu, Jesper Mortensen, and Mel Slater. Presence in response to dynamic visual realism: a preliminary report of an experiment study. In *VRST '06: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 364–367, New York, NY, USA, 2006. ACM.

[Lan09] W. B. Langdon. A fast high quality pseudo random number generator for nvidia cuda. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, pages 2511–2514, New York, NY, USA, 2009. ACM.

[LC04] Bent Dalgaard Larsen and Niels Jorgen Christensen. Simulating photon mapping for real-time applications. In *Rendering Techniques*, pages 123–132, 2004.

[LES09] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. Depth-of-field rendering with multiview synthesis. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pages 134:1–134:6, New York, NY, USA, 2009. ACM.

[LES10] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. Real-time lens blur effects and focus control. *ACM Trans. Graph.*, 29:65:1–65:7, July 2010.

[LH96] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, New York, NY, USA, 1996. ACM Press.

[LHK+04] David Luebke, Mark Harris, Jens Krüger, Tim Purcell, Naga Govindaraju, Ian Buck, Cliff Woolley, and Aaron Lefohn. Gpgpu: general purpose computation on graphics hardware. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes*, page 33, New York, NY, USA, 2004. ACM.

[LHLW09a] Fang Liu, Meng-Cheng Huang, Xue-Hui Liu, and En-Hua Wu. Cuda renderer: a programmable graphics pipeline. In *ACM SIGGRAPH ASIA 2009 Sketches*, SIGGRAPH ASIA '09, pages 34:1–34:1, New York, NY, USA, 2009. ACM.

[LHLW09b] Fang Liu, Meng-Cheng Huang, Xue-Hui Liu, and En-Hua Wu. Efficient depth peeling via bucket sort. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 51–57, New York, NY, USA, 2009. ACM.

[LHLW10] Fang Liu, Meng-Cheng Huang, Xue-Hui Liu, and En-Hua Wu. Freepipe: a programmable parallel rendering architecture for efficient multi-fragment effects. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, I3D '10, pages 75–82, New York, NY, USA, 2010. ACM.

[LSK+07] Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaako Lehtinen, and Timo Aila. Incremental instant radiosity for real-time indirect illumination. In *Eurographics Symposium on Rendering 2007*, June 2007.

[LSSS04] Xinguo Liu, Peter P. Sloan, Heung Y. Shum, and John Snyder. All-frequency precomputed radiance transfer for glossy objects. In Alexander Keller and Henrik W. Jensen, editors, *Eurographics Symposium on Rendering*, pages 337–344, Norrköping, Sweden, 2004. Eurographics Association.

[LTG92] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. A discontinuity meshing for accurate radiosity. *IEEE Comput. Graph. Appl.*, 12(6):25–39, 1992.

[LW93] Eric P. Lafortune and Yves D. Willems. Bi-directional Path Tracing. In H. P. Santo, editor, *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal, 1993.

[LW95] Eric P. Lafortune and Yves D. Willems. A 5d tree to reduce the variance of monte carlo ray tracing. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 11–20, New York, NY, 1995. Springer-Verlag.

[Mam89] Abraham Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Comput. Graph. Appl.*, 9:43–55, July 1989.

[MB07] Kevin Myers and Louis Bavoil. Stencil routed a-buffer. In *ACM SIGGRAPH 2007 sketches*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[MESL10] M. McGuire, E. Enderton, P. Shirley, and D. Luebke. Real-time stochastic rasterization on conventional gpu architectures. In *Proceedings of the Conference on High Performance Graphics*, HPG '10, pages 173–182, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[MGAK03] William R. Mark, R.Steven Glanville, Kurt Akeley, and Mark J. Kilgard. Cg: A system for programming graphics hardware in a c-like language. *ACM Trans. Graph.*, 22(3):896–907, 2003.

[MKYS07a] Jesper Mortensen, Pankaj Khanna, Insu Yu, and Mel Slater. Real-time global illumination in the cave. In *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, pages 145–148, New York, NY, USA, 2007. ACM.

[MKYS07b] Jesper Mortensen, Pankaj Khanna, Insu Yu, and Mel Slater. A visibility field for ray tracing. In *CGIV '07: Proceedings of the Computer Graphics, Imaging and Visualisation*, pages 54–61, Washington, DC, USA, 2007. IEEE Computer Society.

[MM02] Vincent C. H. Ma and Michael D. McCool. Low latency photon mapping using block hashing. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 89–99, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[MSK07] Alexei Soupikov Maxim Shevtsov and Alexander Kapustin. Highly parallel fast kd-tree construction for interactive ray trac- ing of dynamic scenes. In *2007 Eurographics*, 2007.

[MT97] Thomas Mller and Ben Trumbore. Fast, minimum storage ray/triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, 1997.

[MYK+08] Jesper Mortensen, Insu Yu, Pankaj Khanna, Franco Tecchia, Bernhard Spanlang, Giuseppe Marino, and Mel Slater. Real-time global illumination for vr applications. *IEEE Comput. Graph. Appl.*, 28(6):56–64, 2008.

[NC02] Kasper Hoej Nielsen and Niels Joergen Christensen. Fast texture based form factor calculations for radiosity using graphics hardware. *Journal of Graphics Tools*, 6(4):1–12, 2002.

[NPG03] Mangesh Nijasure, Sumanta Pattanaik, and Vineet Goel. Interactive global illumination in dynamic environments using commodity graphics hardware. In *Proc. of 11th Pacific Conference on Computer Graphics and Applications (PG'03)*, 2003.

[NPG05] Mangesh Nijasure, Sumanta Pattanaik, and Vineet Goel. Real-time global illumination on the gpu. *Journal of Graphics Tools*, 10(2):55–71, 2005.

[NPW10] Greg Nichols, Rajeev Penmatsa, and Chris Wyman. Interactive, Multiresolution Image-Space Rendering for Dynamic Area Lighting. *Computer Graphics Forum*, 29(4):1279–1288, 2010.

[NRH+77] F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg, and T. Limperis. Geometric considerations and nomenclature for reflectance. Monograph 161, National Bureau of Standards (US), October 1977.

[NRH03] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. All-frequency shadows using non-linear wavelet lighting approximation. *ACM Trans. Graph.*, 22:376–381, July 2003.

[NW09] Greg Nichols and Chris Wyman. Multiresolution splatting for indirect illumination. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 83–90, New York, NY, USA, 2009. ACM.

[OLG+05] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krger, Aaron E. Lefohn, and Timothy J. Purcell. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, pages 21–51, August 2005.

[PBD+10] Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, and Martin Stich. Optix: a general purpose ray tracing engine. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, pages 66:1–66:13, New York, NY, USA, 2010. ACM.

[PBMH02] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics*, 21(3):703–712, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

[PDC+03] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 41–50, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[Pel95] Marco Pellegrini. Monte carlo approximation of form factors with error bounded a priori. In *Symposium on Computational Geometry*, pages 287–296, 1995.

[Pin88] Juan Pineda. A parallel algorithm for polygon rasterization. *SIGGRAPH Comput. Graph.*, 22:17–20, June 1988.

[PKGH97] Matt Pharr, Craig Kolb, Reid Gershbein, and Pat Hanrahan. Rendering complex scenes with memory-coherent ray tracing. pages 101–108, August 1997.

[PMS+99] Steven Parker, William Martin, Peter-Pike J. Sloan, Peter Shirley, Brian Smits, and Charles Hansen. Interactive ray tracing. pages –, April 1999.

[PO08] Anjul Patney and John D. Owens. Real-time reyes-style adaptive surface subdivision. *ACM Trans. Graph.*, 27:143:1–143:8, December 2008.

[PP98] Ingmar Peter and Georg Pietrek. Importance driven construction of photon maps. In G. Drettakis and N. Max, editors, *Rendering Techniques '98 (Proceedings of Eurographics Rendering Workshop '98)*, pages 269–280, New York, NY, 1998. Springer Wien.

[Pur04] Timothy J. Purcell. *Ray Tracing on a Stream Processor*. PhD thesis, Stanford University, March 2004.

[RCJ98] Erik Reinhard, Alan Chalmers, and Frederik W. Jansen. Overview of parallel photo-realistic graphics. In *Eurographics '98 State of the Art Reports*, pages 1–25. Eurographics Association, August 1998.

[REG+09] T. Ritschel, T. Engelhardt, T. Grosch, H.-P. Seidel, J. Kautz, and C. Dachsbacher. Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph.*, 28(5):1–8, 2009.

[RGK+08] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Trans. Graph.*, 27:129:1–129:8, December 2008.

[RGS09] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, I3D '09, pages 75–82, New York, NY, USA, 2009. ACM.

[SAG94] Brian Smits, James Arvo, and Donald Greenberg. A clustering algorithm for radiosity in complex environments. *Computer Graphics*, 28(Annual Conference Series):435–442, 1994.

[SAS92] Brian E. Smits, James R. Arvo, and David H. Salesin. An importance-driven radiosity algorithm. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 273–282, New York, NY, USA, 1992. ACM Press.

[Sbe93] Mateu Sbert. An integral geometry based method for fast form-factor computation. *Computer Graphics Forum (Eurographics '93)*, 12(3):409–420, 1993.

[Sbe97] Mateu Sbert. *The Use of Global Random Directions to Compute Radiosity: Global Monte Carlo Techniques*. PhD thesis, Barcelona, Spain, 1997.

[SCG97] Peter-Pike J. Sloan, Michael F. Cohen, and Steven J. Gortler. Time critical lumigraph rendering. In *Symposium on Interactive 3D Graphics*, pages 17–24, 181, 1997.

[SD95] Steven M. Seitz and Charles R. Dyer. Physically-valid view synthesis by image interpolation. In *Proc. Workshop on Representation of Visual Scenes*. IEEE Computer Society Press, June 1995.

[SD96] Steven M. Seitz and Charles R. Dyer. View morphing. *Computer Graphics*, 30(Annual Conference Series):21–30, 1996.

[SGHS98] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 231–242, New York, NY, USA, 1998. ACM.

[SGNS07] Peter-Pike Sloan, Naga K. Govindaraju, Derek Nowrouzezahrai, and John Snyder. Image-based proxy accumulation for real-time soft global illumination. In *Proceedings of the 15th*

Pacific Conference on Computer Graphics and Applications, pages 97–105, Washington, DC, USA, 2007. IEEE Computer Society.

[SH92] Robert Siegel and John. Howell. *Thermal Radiation Heat Transfer*, volume 3rd ed. Hemisphere Publishing, New York, 1992.

[Shi90] Peter S. Shirley. A ray tracing method for illumination calculation in diffuse-specular scenes. *Proc.Graphics Interface '90*, pages 205–212, 1990.

[Shi91] Peter S. Shirley. *Physically Based Lighting Calculations for Computer Graphics*. PhD thesis, University of Illinois at Urbana-Champaign, January 1991.

[SKMY09] Mel Slater, Pankaj Khanna, Jesper Mortensen, and Insu Yu. Visual realism enhances realistic response in an immersive virtual environment. *IEEE Comput. Graph. Appl.*, 29(3):76–84, 2009.

[SKS02] P. Sloan, J. Kautz, and J. Snyder. Precomputed radiance transfer for real-time rendering in dynamic, lowfrequency lighting environments. *In Proceedings of SIGGRAPH 2002*, pages 527–536, July, 2002.

[SKT03] Natsuki Sugano, Hirokazu Kato, and Keihachiro Tachibana. The effects of shadow representation of virtual objects in augmented reality. In *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '03, pages 76–, Washington, DC, USA, 2003. IEEE Computer Society.

[Sla02] Mel Slater. Constant time queries on uniformly distributed points on a hemisphere. *Journal of Graphics Tools*, 7(1):33–44, 2002.

[Smi98] Brian Smits. Efficiency issues for ray tracing. *Journal of Graphics Tools: JGT*, 3(2):1–14, 1998.

[SMKY04] Mel Slater, Jesper Mortensen, Pankaj Khanna, and Insu Yu. A virtual light field approach to global illumination. In *CGI '04: Proceedings of the Computer Graphics International*, pages 102–109, Washington, DC, USA, 2004. IEEE Computer Society.

[SP89] François Sillion and Claude Puech. A general two-pass method integrating specular and diffuse reflection. In *Proceedings of SIGGRAPH '89*, pages 335–344. ACM Press, 1989.

[SP94] Franois X. Sillion and Claude Puech. *Radiosity and Global IlluminationRadiosity and Global IlluminationRadiosity and Global Illumination*. Morgan Kaufmann Publishers, Inc., San Francisco, 1994.

[SS92] Kelvin Sung and Peter Shirley. Ray tracing with the bsp tree. In David Kirk, editor, *Graphics Gems III*, volume IBM version, pages 271–274. Morgan Kaufmann Publishers, 1992.

[ST05] Lars Ole Simonsen and Niels Thrane. A comparison of acceleration structures for gpu assisted ray tracing. Master's thesis, University of Aarhus, 2005.

[Std04] The Stanford 3D Scanning Repository. `http://www-graphics.stanford.edu/data/3Dscanrep/`, 2004.

[SW00] Frank Suykens and Yves D. Willems. Density control for photon maps. In B. Peroche and H. Rushmeier, editors, *Rendering Techniques 2000 (Proceedings of the Eleventh Eurographics Workshop on Rendering)*, pages 23–34, New York, NY, 2000. Springer Wien.

[SWZ96] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte carlo techniques for direct lighting calculations. *ACM Transactions on Graphics*, 15(1):1–36, 1996.

[VG94] Eric Veach and Leonidas J. Guibas. Bidirectional estimators for light transport. pages 147–162, July 1994.

[VG95] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 419–428, New York, NY, USA, 1995. ACM Press.

[VG97] Eric Veach and Leonidas J. Guibas. Metropolis light transport. *Computer Graphics*, 31(Annual Conference Series):65–76, 1997.

[WAA+00] Daniel N. Wood, Daniel I. Azuma, Ken Aldinger, Brian Curless, Tom Duchamp, David H. Salesin, and Werner Stuetzle. Surface light fields for 3d photography. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 287–296, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Wal04] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, 2004.

[War91] Gregory Ward. Adaptive shadow testing for ray tracing. In *In 2nd Eurographics Workshop on Rendering*, 1991.

[WBS03] Ingo Wald, Carsten Benthin, and Philipp Slusallek. Interactive global illumination in complex and highly occluded environments. In *Proceedings of the 14th Eurographics workshop on Rendering*, EGRW '03, pages 74–81, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[WBWS01] Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. Interactive rendering with coherent ray tracing. In Alan Chalmers and Theresa-Marie Rhyne, editors, *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2001*, volume 20, pages 153–164. Blackwell Publishers, Oxford, 2001. available at http://graphics.cs.uni-sb.de/ wald/Publications.

[WCG87] John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 311–320, New York, NY, USA, 1987. ACM Press.

[WDP99] Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In D. Lischinski and G.W. Larson, editors, *Rendering techniques '99 (Proceedings of the 10th Eurographics Workshop on Rendering)*, volume 10, pages 235–246, New York, NY, Jun 1999. Springer-Verlag/Wien.

[WEH89] John R. Wallace, Kells A. Elmquist, and Eric A. Haines. A ray tracing algorithm for progressive radiosity. *Computer Graphics*, 23(3):315–324, July 1989.

[WFA+05] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: a scalable approach to illumination. *ACM Trans. Graph.*, 24:1098–1107, July 2005.

[WGER05] Daniel Wexler, Larry Gritz, Eric Enderton, and Jonathan Rice. Gpu-accelerated high-quality hidden surface removal. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '05, pages 7–14, New York, NY, USA, 2005. ACM.

[WH92] Gregory J. Ward and Paul Heckbert. Irradiance gradients. In *Third Eurographics Workshop on Rendering*, pages 85–98, Bristol, UK, 1992.

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, 1980.

[Woo90] Andrew Woo. Fast ray-polygon intersection. In *In Andrew S. Glassner, editor, Graphics Gems*, page 394. Academic Press, San Diego, 1990.

[WRC88] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 85–92, New York, NY, USA, 1988. ACM Press.

[WS99] Gregory Ward and Maryann Simmons. The holodeck ray cache: an interactive rendering system for global illumination in nondiffuse environments. *ACM Trans. Graph.*, 18(4):361–368, 1999.

[WS03] Michael Wand and Wolfgang Straßer. Real-time caustics. In P. Brunet and D. Fellner, editors, *Computer Graphics Forum*, volume 22(3), 2003.

[WSC+95] Kyu-Young Whang, Ju-Won Song, Ji-Woong Chang, Ji-Yun Kim, Wan-Sup Cho, Chong-Mok Park, and Il-Yeol Song. Octree-r: An adaptive octree for efficient ray tracing. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):343–349, 1995.

[WSS05] Sven Woop, Jörg Schmittler, and Philipp Slusallek. Rpu: A programmable ray processing unit for realtime ray tracing. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 434–444, New York, NY, USA, 2005. ACM.

[WWZ+09] Rui Wang, Rui Wang, Kun Zhou, Minghao Pan, and Hujun Bao. An efficient gpu-based approach for interactive global illumination. In *ACM SIGGRAPH 2009 papers*, SIGGRAPH '09, pages 91:1–91:8, New York, NY, USA, 2009. ACM.

[YCK+09] Insu Yu, Andrew Cox, Min H. Kim, Tobias Ritschel, Thorsten Grosch, Carsten Dachsbacher, and Jan Kautz. Perceptual influence of approximate visibility in indirect illumination. In *APGV '09: Proceedings of the 6th Symposium on Applied Perception in Graphics and Visualization*, New York, NY, USA, 2009. ACM.

[YMKS11] Insu Yu, Jesper Mortensen, Pankaj Khanna, and Mel Slater. Visual realism enhances realistic response in an immersive virtual environment - part 2 (to appear). *IEEE Comput. Graph. Appl.*, 2011.

[ZHR+09] Kun Zhou, Qiming Hou, Zhong Ren, Minmin Gong, Xin Sun, and Baining Guo. Renderants: interactive reyes rendering on gpus. *ACM Trans. Graph.*, 28:155:1–155:11, December 2009.

[ZHWG08] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time kd-tree construction on graphics hardware. *ACM Trans. Graph.*, 27:126:1–126:11, December 2008.

[ZIK98] Sergey Zhukov, Andrei Iones, and Grigorij Kronin. An ambient light illumination model. In *Eurographics Symposium on Rendering/Eurographics Workshop on Rendering Techniques*, pages 45–56, 1998.

[ZS95] Kurt Zimmerman and Peter Shirley. A two-pass realistic image synthesis method for complex scenes. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 284–295, New York, NY, 1995. Springer-Verlag.